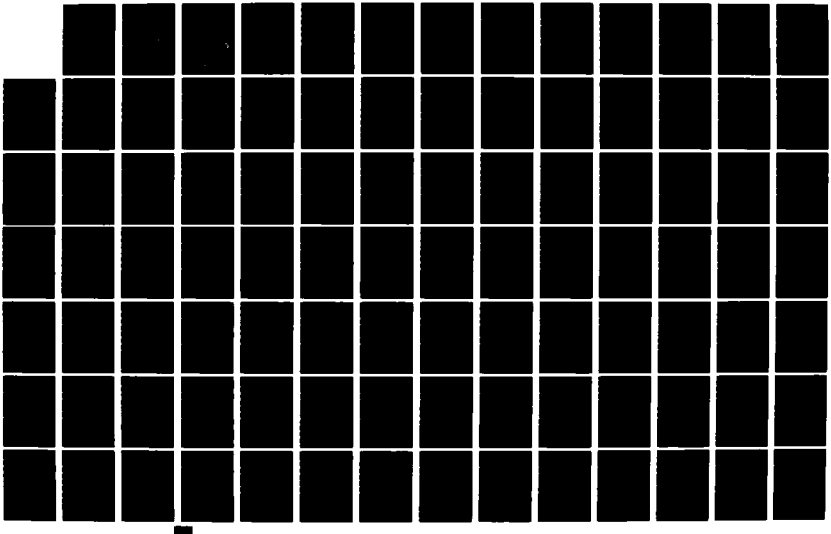
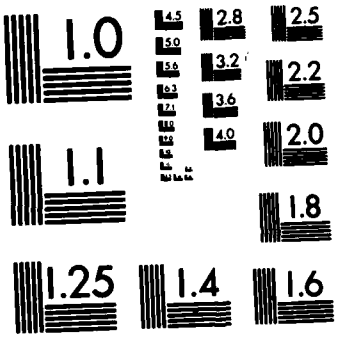
AD-A124 921 QUERY OPTIMIZATION IN DISTRIBUTED DATABASES(U) 1/3
MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR
INFORMATION AND DECISION SYSTEMS K HUANG OCT 82
UNCLASSIFIED LIDS-TH-1247 N00014-77-C-0532 . F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

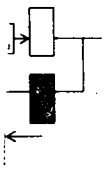
(12)

October, 1982

LIDS-TH-1247

Research Supported By:

Office of Naval Research
Contract ONR/N00014-77-C-0532
(NR 041-519)



QUERY OPTIMIZATION IN DISTRIBUTED DATABASES

Kuan-Tsae Huang

Laboratory for Information and Decision Systems
MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MASSACHUSETTS 02139

(12)

October, 1982 LIDS-TH-1247

QUERY OPTIMIZATION IN DISTRIBUTED DATABASES

by

Kuan-Tsae Huang

This report is based on the unaltered thesis of Kuan-Tsae Huang, submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at the Massachusetts Institute of Technology in September, 1982. The research was conducted at the M.I.T. Laboratory for Information and Decision Systems, with support provided by the Office of Naval Research under contract ONR/N00014-77-C-0532 (NR 041-519).

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139

Query Optimization in Distributed Databases

by

Kuan-Tsae Huang

B.S., National Taiwan Normal University
(1974)

M.S., University of Illinois at Urbana-Champaign
(1979)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS OF THE
DEGREE OF

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1982

Signature of Author Kuan-Tsae Huang
Department of Electrical Engineering
and Computer Science, Sep. 1982

Certified by William B. Davenport Jr
Thesis Supervisor

Accepted by
Chairman, Departmental Committee on Graduate Students

Query Optimization in Distributed Databases

by

Kuan-Tsae Huang

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

ABSTRACT

Distributed database management systems (DDBMS) are amongst the most important and successful software developments in this decade. They are enabling the computing power and data to be placed within the user environment close to the point of user activities. The performance efficiency of DDBMS is deeply related to the query processing strategies involving data transmission over different nodes through the network. This thesis is to study the optimization of query processing strategies in a distributed databases environment.

With the objective of minimum communication cost, we have developed a mathematical model to find a join-semijoin program for processing a given equi-join query in distributed homogenenous relational databases. Rules for estimating the size of the derived relation is proposed. The distributed query processing problem is formulated as dynamic network problem. We also extend this model to consider both communication cost and local processing cost. For a simpler case where all semijoin reducibilities are zero and semijoin reducibilities do not change after join operation, we have shown that under three different objective functions, the problems of finding a routing strategy of required data to the site where a query is initiated are NP-complete. We analyze the difficult nature of the query processing problem and provide an analytical basis for heuristic algorithms.

We extend this model to query processing in a distributed heterogeneous databases environment. A heterogeneous database communication system is proposed to integrate heterogeneous database management systems to combine and share information. The use of a database communication system for heterogeneous DBMSs makes the overall system transparent to users from an operational point of view. Problems of schema translation and query translation of the query processing in this environment are studied.

Supervisor: Dr. Wilbur B. Davenport, Jr.

Title: Professor of Communications Science and Engineering

ACKNOWLEDGMENTS

I would first of all like to express my gratitude to my thesis supervisor, Professor Wilbur B. Davenport, Jr. He has been a constant resource of encouragement, guidance and support throughout my graduate studies. His patient effort in correcting my writing greatly improve this thesis. I also like to thank to my thesis readers, Professors Robert Gallager, Stuart Madnick and Christos Papadimitriou for many constructive suggestions to improve this thesis. Espicially, the weekly discussion with Christos in the first stage of the thesis and the supervision from Bob at the last stage of the thesis are very helpful. Special thanks go to them.

During my stay at M.I.T., I have benefited immensely from Professors Dimitri Bersekas, Ravi Kannan, Richard Larson and Thomas Magnatti for their support and advise.

I would also like to thank my friends from Taiwan, my fellow colleagues from Operations Research Center and Laboratory for Information and Decision Systems at M.I.T. for their friendship, moral and intellectual support.

My deepest appreciation goes to my wife Lu-Yue by my side, my parents, brothers and sister at Taiwan, who have unreservedly given me their love, support and understanding. I shall be forever thankful for them.

This research was conducted at the M.I.T. Laboratory for Information and Decision systems with support provided by the Office of Naval Research under contract ONR/N00014-77-C-0532 (NR 041-519).

DEDICATION

To My Parents

Ching-Ming and Jen-Uyeh (Chang) Huang

June, 1982

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 Background and Motivation

Database management systems are amongst the most important and successful software developments in this decade. They have already had a significant impact in the field of data processing and information retrieval. Many organizations have successfully developed their own database management systems for storing and accessing information to help their operations. The increaing geographic dispersion of the business activities within an organization forces the enterprise to develop a distibuted database management system (DDBMS) in order to provide its users faster and easier access to data for decision making as well as to keep the system reliable and secure. For example, in militay Command, Control and Communication (C^3) systems, data gathering from sensors and commanders are distributed in nature. A centralized database management system can not provide the availability, reliability and modularity that is needed.

The development of computer networks since 1970, and the emergence of low-cost, yet powerful, small mini and micro computers makes it possible to develop a distributed database management system enabling the computing power and data to be placed within the user environment close to the point of user activities. The development of DDBMS are

apparently the result of the convergence of these different technologies. It certainly will make possible both economical and technical advantages such as faster access to data, better performance, increased reliability, easier upward expansion, and more information sharing, etc.

However, the benefits of DDBMS can not be gained without cost. Several problems inherent to distributed systems must be solved as must other problems related to database systems in general. These additional distributed system problems come from the slow transmission speed, narrow bandwidth and possibly high failure rate of the communication channels. The speed of communication transmissions are slow compared to the CPU, main memory and secondary storage times. These distributed system problems include concurrency control, recovery, database integrity, query processing, directory management, security, etc.

Recent studies [ESW 78, GBWRR 81, HY 79, DL 80, DP 80] on distributed databases have shown that the performance efficiency of DDBMS is deeply related to the query processing strategies involving data transmission over different nodes through the network. Moreover, the communication technologies have not yet reached the same level as the computer technologies, both in cost and performance, and it is expected that this situation will not be changed in the near future. So, in this thesis we study the optimization of query processing strategies in a

distributed databases environment.

1.2 Assumptions and Objectives

Query Processing in a distributed database environment corresponds to the translation of a request formulated in a high-level nonprocedural language on one sub-system of the network, into a sequence of data manipulation statements which retrieve and update data stored in local DBMSs. The objective of this thesis is to develop a quantitative and syntatic understanding of the optimization of query processing strategies in a distributed database management systems environment. Particular emphasis will be given to the minimization of the total amount of data transmission cost required for processing a single query.

This thesis starts with a mathematical model for equi-join query processing in distributed relational database management systems. It then studies the computational complexity of this problem and some solution algorithms. The query processing in a distributed heterogeneous database environment is also studied.

Our basic underlying assumptions in this thesis are:

1. It is possible to exchange information amongst the various systems and they are willing to maintain information.
2. Each DBMS is considered to be able to execute a given local transaction.

3. There exists a communication network which connects the various DBMSs.
4. The access to a local DBMS is not affected by the operation of the data communication system which should be transparent to the local user.
5. The communication cost is the dominant factor and local processing is essentially free.

As we are only concerned with distributed query processing in this thesis, we also assume that other problems related to distributed database management systems, such as concurrency control, database integrity, redundancy, recovery and security poblems have been taken care of by some other components of the system.

1.3 Organization of The Thesis

This thesis is organized into seven chapters.

Chapter 1 contains the background and motivation of this thesis. Query processing in distributed databases is the main theme of this thesis. Assumptions about distributed database environments are stated and objectives of this thesis are defined.

Chapter 2 discusses query processing in a distributed relational database management system environment. As communication technologies have not yet reached the same level of reduction in costs and of increases in performance that we can observe in computer technologies, we assume

network traffic constitutes the critical factor. We formalize the problem of solving an equi-join query by joins and semijoins mixed strategies into a mathematical model. Query processing for a broader class of query is also discussed.

Chapter 3 considers the computational complexity of the query processing problem formulated in chapter 2. Three variations of simpler cases of the query processing problem with different objective functions are studied.

Chapter 4 focuses on heuristic algorithms for the query processing problem. We first consider the heuristic algorithms for the cases where all possible semijoins are performed and find a routing strategy for the join operations to solve the query. Next, we consider heuristic algorithms for the general case where we want to solve a given query by a sequence of joins and semijoins.

Chapter 5 extends the previous results on distributed relational databases to the query processing in distributed heterogeneous databases. It begins with a study of the heterogeneous world of database management systems. The architecture of a heterogeneous database communication system is described to integrate those heterogeneous, distributed and nonintegrated database management systems. Query processing in heterogeneous database environments by using a database communication system is discussed.

Chapter 6 deals with two essential components of a database communication system for query processing: schema translators to translate from the data model schema of the database management system to relational schema in order to provide a uniform relational view to the user; and query translators to translate relational algebra operations into corresponding data manipulation statements of the underlying data model of the system in order to retrieve data.

Chapter 7 provides a summary of the major results of each chapter and suggests several potential areas for future research.

Chapter 2

Query Processing in Distributed Relational Databases

2.1 Introduction

Query processing in a Distributed Relational Data Base Management System (DDBMS) corresponds to the translation of requests, formulated in a high-level language on one system of the network, into a sequence of relational algebra operations which retrieve and update data stored in the DDBMS. The optimization process is usually subdivided into two parts: The first part is used at the node where the query is generated for analyzing the query before executing it and producing as output a sequence of operational commands to the local DBMS, which is optimal in terms of some cost function criteria. The second part is used by the local DBMSs for further analyzing each operational command and producing the optimal local data retrieval strategy. The result of these operations constitutes the final response returned to the user.

In distributed query processing, the execution of a query involves data transmissions which take significant time in comparison with the subquery and elementary operation execution times. However, on the other hand, the distribution of the system makes possible the parallel processing of local elementary operations, which is beneficial. The cost function of processing a query in a

DDBMS environment depends on the following parameters:

1. Total transmission time.
2. Total execution time.
3. Total resources usage.
4. Total response time.

Some of the parameters are dependent on others, therefore the cost function will depend on all four parameters and may not be linear. In a distributed DBMS, the main bottle-necks of the system are the transmission delays of the inter-computer communications. The cost function may be simplified by considering only the quantity of data transmitted through the network. Most of the query processing algorithms proposed to date concern only the reduction of the total transmission time.

In this chapter, we first formalize some definitions for distributed relational DBMS and derive some basic results in section 2.2. In section 2.3, we review the literature of distributed query processing algorithms. We extend previous work in two directions: One is that we want to solve a distributed query by using a join and semijoin mixed strategy. The other is that we want to formalize the problem in a mathematical formulation. In section 2.4, we develop a distributed query processing model for a class of conjunctive equi-join queries and formulate the problem as a dynamic network problem. In section 2.5, we extend the model to general query processing where inequality-join

clauses could appear. The model can also be extended to include the local processing cost into consideration.

2.2 Relational Terminology and Basic Lemmas

In this section, we formalize the time-invariant description of a relation and a database.

2.2.1 Schema

A domain is a set of values. A relation is any subset of the cartesian product of one or more domains. The members of a relation are called tuples. A tuple $t=(t_1, t_2, \ldots, t_n)$ has n components; the i-th component of t is t_i. We can view a relation as a table, where each row is a tuple and each column corresponds to one component. The columns are often given names, called attributes. Let U be a set of attributes, called a Universe. Associated with each attribute $A \in U$ is a domain, DOM(A). A relation schema R is a list of attribute names for a relation, i.e. a subset of U. Let $R_i = \{A_1, A_2, \ldots, A_m\}$ be a relation schema. We will use the notation $R_i.A_k$ to denote the attribute A_k of relation R_i. The domain of R_i is $DOM(R_i) = \overset{m}{\underset{j=1}{X}} DOM(R_i.A_j)$. Let R_1, R_2, ..., R_n be relation schemas. A database schema D is defined as a set of relation schemas $\{R_1, R_2, \ldots, R_n\}$. U(D) is the attribute set of D, i.e. $U(D) = \overset{n}{\underset{i=1}{U}} R_i$. The domain of D is denoted DOM(D) and is defined to be $\overset{n}{\underset{i=1}{X}} DOM(R_i)$. Let D_1, $D_2, \ldots, D_p$ be a set of database schemas. Each D_k corresponds

to a database at one location. A distributed database schema DD is defined as a set of database schemas $\{D_1, D_2, \ldots, D_p\}$ and U(DD) is the set of attributes in DD.

When set notation is used (e.g. $R_i \cup R_j$, $A_K \in R_i$, $R_i \subseteq R_j$), the sets in question are understood to be sets of attributes. $R_i = R_j$ iff $R_i \subseteq R_j$ and $R_j \subseteq R_i$, i.e. R_i and R_j are two relation schemas having the same set of attributes. Two database schemas, $D = \{R_1, R_2, \ldots, R_n\}$ and $D = \{S_1, S_2, \ldots, S_n\}$, are equivalent if there exists a one-to-one correspondence between their relations with respect to the equality of two relation schemas, i.e. there exists a n-permutation σ such that $R_i = S_{\sigma(i)}$, for all i=1,2,...,n.

Definition:

We define as follows the size of a database schema:

The size of a distributed database schema DD, $|DD|_n$, is the number of different sites in DD.

The relation size of a database schema D_i, $|D_i|_r$, is the number of relation schemas in D.

The relation size of a distributed database schema DD, $|DD|_r$, is equal to

$$\sum_{i=1}^{|DD|_n} |D_i|_r .$$

The attribute size of a relation schema R_i, $|R_i|_a$, is the number of attributes in R_i.

The attribute size of a database schema D , $|D|_a$ is equal to $|U(D)|$, where $U(D)=\bigcup_{i=1}^{n} U(R_i)$.

DEFINITION:

D_i is a database subschema of D_j , denoted by $D_i \leq D_j$, if for all $R_K \in D_i$, there exists $R_l \in D_j$ such that $R_K \subseteq R_l$.

$R_K \in D_i$ is maximal if there exists no $R_l \in D_i$ such that $R_K \subseteq R_l$.

DEFINITION:

For each D_i , $A \in U(D_i)$, we define $s(A,D_i)=\{R_K | (R_K \in D_i)$ and $(A \in R_K)\}$, the set of all relation schemas in D_i for which A is an attribute.

For $A \in U(DD)$, we define $s(A)=\{D_i | (D_i \in DD)$ and $(|s(A,D_i)| \geq 1)\}$

DEFINITION:

Attribute A is said to be isolated in DD if $|s(A)|=1$. i.e. if A is an attribute of only one D_i.

DEFINITION:

A schema graph for a database schema $D=\{R_1, R_2, \ldots, R_n\}$ is a graph $G_D=<V_D, E_D>$ with node set $V_D=D$ and edge$(R_i, R_j) \in E_D$ iff $U(R_i) \cap U(R_j) \neq \phi$.

A schema graph for distributed database schema $DD=\{D_1, D_2, \ldots, D_p\}$ is a graph $G_{DD}=<V_{DD}, E_{DD}>$ with node set $V_{DD}=DD$ and edge $(D_i, D_j) \in E_{DD}$ iff $U(D_i) \cap U(D_j) \neq \phi$.

2.2.2 Database state

Let $R=\{A_1, A_2, .., A_m\}$ be a relation schema. A tuple T over R is a mapping which associates with each attribute a value out of a distinct domain associated with the attribute. When the order of attributes in R is fixed, we may write T as an m-tuple $T=(T(A_1), T(A_2), ..., T(A_m))$.

Definition:

1. A relation state r of R is a finite set of tuples over R. r may be visualized as a table of data whose columns are labeled A_1, A_2, ..., A_m and whose rows are tuples.
2. If $D=\{R_1, R_2, ..., R_n\}$, then a database state for D is $d=\{r_1, r_2, ..., r_n\}$ where for $1 \leq i \leq n$, r_i is a relation state for R_i ; we denote the database state by d.
3. If $DD=\{D_1, D_2, ... D_p\}$ then a distributed database state for DD is $dd=\{d_1, d_2, ..., d_p\}$ where for $1 \leq k \leq m$, d_k is a database state for D_k , we denote the distributed database state by dd.

Note that, if there is no confusion, we will use capital letters (e.g. R, D, DD) to represent both schema and state through out the thesis. For example, relation R=(S,r) is to represent both its relation schema S and relation state r.

Example: Relations and Databases

(1) Three relation schemas:

CLIENT={CNUMBER, NAME, ADDR, AGE, BIRTHDAY}

AGENT= {ANUMBER, NAME, ADDR, TERRITORY, SENIORITY}

SALE= {ANUMBER, CNUMBER, POLICYNO, DATE}

(2) Database schema:

D = { CLIENT, AGENT, SALE }

(3) Set of attributes:

U(CLIENT)={CNUMBER, NAME, ADDR, AGE, BIRTHDAY}

U(D)={CNUMBER, NAME, ADDR, AGE, BIRTHDAY, ANUMBER, TERRITORY, SENIORITY, POLICYNO, DATE}

(4) Relation states and database state:

CLIENT

CNUMBER	NAME	ADDR	AGE	BIRTHDAY
11	TOTO	MAIN ST.	32	12/18
12	YOYO	WEST ST.	41	3/14
13	LILI	EAST ST.	26	6/30
14	PAPA	NORTH ST.	29	5/11
15	DADA	MAIN ST.	30	7/31
16	QUQU	EAST ST.	37	9/17

AGENT

ANUMBER	NAME	ADDR	TERRITORY	SENIORITY
A1	JOHN	MAIN ST.	CENTER	10
A2	TOM	WEST ST.	WEST	8
A3	PAUL	NORTH ST.	NORTH	5
A4	DICK	SOUTH ST.	SOUTH	3
A5	JEFF	EAST ST.	EAST	11

SALE

ANUMBER	CNUMBER	POLICYNO	DATE
A1	11	3	4/78
A2	12	4	5/80
A5	13	7	9/80
A3	14	1	9/81
A1	15	4	12/81
A5	16	1	12/81

2.2.3 Operator

The operators we use are a subset of the relational algebra [CODD 72].

Definition:

Let $R=\{A_1, A_2, \ldots, A_m\}$, and let r be the relation state of R and $t=(t_1, t_2, \ldots, t_m)$ be a tuple in r. The projection of tuple t on attributes $Y=\{A_{i1}, A_{i2}, \ldots, A_{im}\} \subset R$ is $t[Y]=(t_{i1}, t_{i2}, \ldots, t_{im})$. That is, the projection is obtained by removing from t those components corresponding to attributes not in Y.

The projection of relation state r onto an attribute set Y is $r[Y]=\{s' \mid (s'=s[Y])$ and $(s$ is a tuple of $r)\}$. Also denoted as $\pi_Y[r]$.

Definition:

Let $R_1=(S_1, r_1)$ and $R_2=(S_2, r_2)$ be two relations with the same relation schemas, i.e. $S_1 = S_2$. The intersection of these two relations, denoted by $R_1 \cap R_2$, is the relation (S,r), where $S = S_1 = S_2$ is the relation schema and $r=\{t \mid t \in r_1 \cap r_2\}$ is the relation state.

Definition:

Let R and S be relations of arity (number of attributes) h and k, respectively. The Cartesian product of R and S, denoted by R x S is the set of (h+k)-tuples whose first h components form a tuple in R and whose last k components form a tuple in S.

Definition:

Let $\{R_i\}_{i=1}^{n}$ be a set of relations and the schema of relation R_i be the set of attributes $\{A_{i1}, A_{i2}, \ldots, A_{im_i}\}$. A qualification q is a formula of clauses of the forms $(R_i.A_{ie}\ \theta\ R_j.A_{jf})$ and $(R_i.A_{ie}\ \theta\ C)$, where θ is an arithmetic operator ($<$, $=$, $>$, $\leq$, $\geq$ or $\neq$) and C is a constant value in the domain of the attribute A_{ie} of R_i, connected by logical operators ($\wedge$, $\vee$, and $\neg$).

Definition:

The selection of relation R on qualification q,

denoted $\sigma_q(R) = \{t=(t_1, t_2, \ldots, t_n) \in R \mid q\ (t_1, t_2, \ldots, t_n)$ is true$\}$. i.e. σ_q (R) is the set of tuples in R that satisfy q.

Let R_i and R_j be two relations and $Y \subseteq R_i \cap R_j$.

Definition:

The equi-join of R_i and R_j on Y, denoted by $R_i \ |\underset{Y}{X}|\ R_j$, is the resulting relation of computing $\{t \mid t$ is a tuple in $R_i \times R_j$, such that $t[R_i.Y] = t[R_j.Y]\}$ and then projecting out columns $R_j.Y$.

The semijoin of R_i and R_j on Y, denoted by $R_i \ |\underset{Y}{X}\ R_j$, equals $R_i \ |\underset{Y}{X}|\ R_j[Y]$. Equivalently it equals $\{t_i \mid (t_i \in r_i)$ and $(\exists\, t_j \in r_j \ni t_i[Y]=t_j[Y])\}$.

The natural join of R_i & R_j, denoted by $R_i \ |X|\ R_j$, is the join of R_i & R_j on $R_i \cap R_j$.

The natural semijoin of R_i & R_j, denoted by $R_i \ |X\ R_j$, is the semijoin of R_i & R_j on $R_i \cap R_j$.

Note that the natural join is commutative. i.e.

$$R_i \ |X|\ R_j = R_j \ |X|\ R_i .$$

Definition:

The natural join of $R_i = (S_i, r_i)$, $R_j = (S_j, r_j)$ and $R_k = (S_k, r_k)$, denoted by $R_i \ |X| R_j \ |X| R_k$, is the resulting relation of $(R_i \ |\underset{Y_{ij}}{X}| R_j) \ |\underset{Y_{ik} \cup Y_{jk}}{X}|\ R_k$. Here $Y_{ij} = S_i \cap S_j$ and $Y_{ik} \cup Y_{jk} = (S_i \cup S_j) \cap S_k$.

Lemma: For natural joins, we have

$$(R_i \underset{Y_{ij}}{|X|} R_j) \underset{Y_{ik} \cup Y_{jk}}{|X|} R_k = R_i \underset{Y_{ij} \cup Y_{ik}}{|X|} (R_j \underset{Y_{jk}}{|X|} R_k).$$

Proof:

1. The relation schemas of both resulting relations above are the same, which is $S_i \cup S_j \cup S_k$.

2. Let r_1 be the state of $(R_i \underset{Y_{ij}}{|X|} R_j) \underset{Y_{ik} \cup Y_{jk}}{|X|} R_k$, r_2 be the state of $R_i \underset{Y_{ij} \cup Y_{ik}}{|X|} (R_j \underset{Y_{jk}}{|X|} R_k)$ and r_3 be the state of $R_i \underset{Y_{ij}}{|X|} R_j$. We can derive

$$r_1 = \{ t \mid \exists t_k \in r_k \ \& \ t_0 \in R_i |X| R_j \ \text{ s.t. } t[R_k] = t_k, \ t[R_i \cup R_j] = t_0 \}$$

$$= \{ t \mid \exists t_k \in r_k, t_i \in r_i, t_j \in r_j \ \& \ t_0 \in R_i |X| R_j \ \text{ s.t. } t[R_k] = t_k, \ t[R_i] = t_0[R_i] = t_i \text{ and } t[R_j] = t_0[R_j] = t_j \}$$

$$= \{ t \mid \exists t_k \in r_k, t_i \in r_i, t_j \in r_j \ \text{ s.t. } t[R_k] = t_k, \ t[R_i] = t_i \text{ and } t[R_j] = t_j \} \quad (*)$$

Similary, we can derive r_2 = the formula (*). Thus $r_1 = r_2$, and the lemma follows.

Note that the natural join operation is associative, i.e.

$$(R_i |X| R_j) |X| R_k = R_i |X| (R_j |X| R_k).$$

Example:

(1) Join $R_1 \underset{Y}{|X|} R_2$

R_1

A_1	A_2	A_3
11	21	31
12	22	31
13	21	32
14	22	32

R_2

A_2	A_3	A_4
21	31	41
22	32	42
21	33	43
22	34	41
21	35	42

$R_1 \underset{\{A_3\}}{|X|} R_2$

A_1	A_2	A_3	A_2'	A_4
11	21	31	21	41
12	22	31	21	41
13	21	32	22	42
14	22	32	22	42

$R_1 \underset{\{A_2,A_3\}}{|X|} R_2$

A_1	A_2	A_3	A_4
11	21	31	41
14	22	32	42

This is the natural join since $\{A_2, A_3\} = R_1 \cap R_2$.

(2) Semijoin $R_1 \underset{Y}{X|} R_2$

$R_1 |X|_{\{A_3\}} R_2$ is to join

A_3
31
32

with R_2 =

A_2	A_3	A_4
21	31	41
22	32	42

$R_1 |X|_{\{A_2,A_3\}} R_2$ is to join

A_2	A_3
21	31
22	31
21	32
22	32

with R_2 =

A_2	A_3	A_4
21	31	41
22	32	42

This is a natural semijoin since $\{A_2,A_3\}=R_1 \cap R_2$.

(3) $R_1 |X| R_2 |X| R_3$

R_3 =

A_3	A_4	A_5
31	41	51
31	43	53
32	44	52
32	44	51

Then $R_1 |X| R_2 |X| R_3 = (R_1 |X|_{\{A_2,A_3\}} R_2) |X|_{\{A_3,A_4\}} R_3$.

=

A_1	A_2	A_3	A_4	A_5
11	21	31	41	51

2.2.4 Query

Given a database $D=\{R_1,R_2,\ldots,R_n\}$, a query can usually be written in a number of alternative algebraic expressions.

In particular each query can be put in the following form:

$$Q = \Pi_{TL}\, \sigma_{q}(R_1 x R_2 x \ldots x R_n)$$

where TL contains the attributes in the answer relation; q is a qualification and each R_i is a relation. Usually, TL is referred to as the target-list. We shall assume all queries are expressed in this canonical form, denoted by Q=(q, TL). A query Q is called single relational (SR) if TL is a subset of some relation schema in D, otherwise, Q is multirelational (MR).

Definition:

> A query Q=(q, TL) is a conjunctive equi-join query if the qualification q is a conjunction of equi-join clauses of the form (R_i. X= R_j.Y), where X and Y are subsets of attributes of R_i and R_j respectively.

Note that equi-join queries do not include one-relation clauses of the form (R_j.Y =constant). One-relation clauses were excluded because they correspond to local operations and are generally evaluated locally before join and semijoin operations are applied.

Let DD={R_1, R_2,..., R_p} be a distributed database schema and Q=(q,TL) be a query over DD. For every distributed database state dd, the answer of Q over (DD,dd), denoted by Q(dd), is $Q(dd) = \Pi_{TL}\, \sigma_{q}(r_1 x r_2 x \ldots x r_p)$.

Definition: (equivalence)

Given a database schema DD, two queries Q_1 and Q_2 are equivalent, denoted $Q_1 \equiv Q_2$, iff for all database states dd, $Q_1(dd)=Q_2(dd)$.

Definition:

Let Q=(q,TL) be a query. The transitive closure of Q is a query $Q^+ = (q^+, TL)$ whose qualification q^+, includes q and all clauses implied by q under transitivity. (e.g. If $(R_1.A_1=R_2.A_2)$ and $(R_2.A_2=R_3.A_3)$ are in q, then $(R_1.A_1=R_3.A_3)$ is in q^+).

Lemma:

Given a database schema, a query Q is equivalent to its closure Q^+. i.e. $Q \equiv Q^+$.

Proof:

Let Q=(q,TL) and $Q^+=(q^+, TL)$. By the definition of q^+, for each database state $dd=\{r_1, r_2, \ldots, r_n\}$, any tuple t in $r_1 x r_2 x \ldots x r_n$ that satisfies q satisfies every clause in q. Since every clause in q^+ is either a clause in q or a clause derived from clauses in q by transitivity, the tuple t also satisfies every clause in q^+. The other direction is also true. So $\sigma_q(r_1 \times r_2 \times \ldots \times r_n) = \sigma_{q^+}(r_1 \times r_2 \times \ldots \times r_n)$. Also the target list of Q and Q are the same. This implies $Q(dd)=Q^+(dd)$. Thus $Q \equiv Q^+$.

Definition:

A qualification q is called sub-natural iff for each clause $R_i.A_{iK}=R_j.A_{jL}$, $A_{iK}=A_{jL}$.

A qualification q is called natural iff q is a subnatural qualification and for all relation schemas R_i and R_j, and for all $A_k \in R_i \cap R_j$, $R_i.A_k = R_j.A_k$ is a clause of q.

Definition:

Given a database schema $D=\{R_1, R_2, \ldots, R_n\}$, a query $Q=(q,TL)$ is called a natural join query (NJQ) (resp. sub-natural join query , SNJQ) iff q is a natural qualification (resp. sub-natural qualification) for it and $TL \subseteq U(D)$.

2.2.5 Qual Graph

Next we define qual graphs for the class of sub-natural join (SNJ) qualifications. Qual graphs are another structure for specifying sub-natural queries. Let D be a database schema and let $U(D) = \bigcup_{i=1}^{n} R_i$.

Definition:

A qual graph for $Q=(q,TL)$ over D is an edge-labelled undirected graph $G_Q = \langle V_Q, E_Q, L_Q \rangle$, where V_Q contains one node per relation schema in D, $E_Q = V_Q \times V_Q$, and $L_Q : E_Q \rightarrow 2^{U(D)}$, where $2^{U(D)}$ is the set of all subsets of U(D), with $L[(R_i,R_j)] = \{A_k \in U(D) \mid R_i.A_k = R_j.A_k$ is a clause in q$\}$.

Note that:

1. The qual graph of $Q=(q,TL)$ is uniquely defined by the qualification q.
2. The label of edge (R_i,R_j), $L_q\{(R_i,R_j)\}$ may be the

empty set ϕ.

Definition:

The transitive closure of $G_Q = < V_Q, E_Q, L_Q >$, denoted by $G_Q^+ = < V_Q^+, E_Q^+, L_Q^+ >$ is $V_Q^+ = V_Q$, $E_Q^+ = E_Q$, and

$L_Q^+\{(R_i, R_j)\} = \{ A_k \in U(D) \mid$ there is a path from R_i to R_j in G_Q such that A_k is in the label of each edge on the path $\}$.

Lemma 1:

Let $Q=(q,TL)$ be a sub-natural join graph and its qual graph be $G_Q = <V_Q, E_Q, L_Q>$, then $G_Q^+ = G_{Q^+}$.

Lemma 2:

Let $Q=(q,TL)$ be a natural join query then $G_Q = G_Q^+ = G_{Q^+}$

.

Lemma 3:

Let $Q_1=(q_1, TL_1)$ and $Q_2=(q_2, TL_2)$ be sub-natural join queries. $Q_1 \equiv Q_2$ iff $G_{Q_1}^+ = G_{Q_2}^+$ and $TL_1 = TL_2$.

In the sequel we will only consider sub-natural join queries whose qual graphs are connected. A query whose qual graph is disconnected produces a result that is the Cartesian product of database substates produced by each connected component. Since these connected components are not joined in any way, there is no loss of generality in treating the components seperately.

Definition:

A sub-natural join query $Q=(q,TL)$ is called a tree query iff there exists a connected qual graph $G_{Q'}$ such that $Q' \equiv Q$ and $G_{Q'}$ is a tree. All other sub-natural join queries with connected qual graphs which do not have equivalent queries with tree qual graphs are called cyclic queries.

Example:

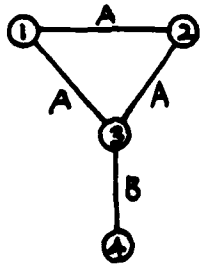
Let $R_1=\{A,C\}$, $R_2=\{A,D\}$, $R_3=\{A,B\}$ and $R_4=\{B,D,E\}$.

1. $Q_1=(q_1,TL)$, where $q_1 = (R_1.A = R_2.A) \wedge (R_2.A = R_3.A) \wedge (R_1.A = R_3.A) \wedge (R_3.B = R_4.B)$

Q_1 is a subnatural join query.

The qual graph G_{Q_1} of Q_1 is not a tree.

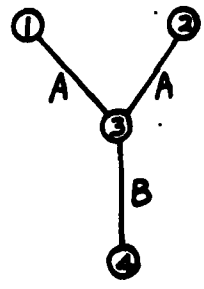
G_{Q_1}



$Q_2=(q_2,TL)$, where $q_2 = (R_1.A = R_3.A) \wedge (R_2.A = R_3.A) \wedge (R_3.A = R_4.A)$

The qual graph G_{Q_2} of Q_2 is a tree.

G_{Q_2}

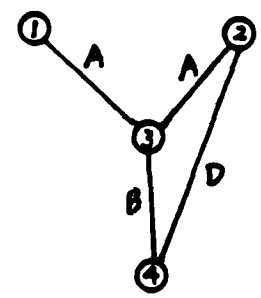


Thus, Q_2 is a tree query because $G^+_{Q_1} = G^+_{Q_2}$ (i.e. $Q_1 \equiv Q_2$), and G_{Q_2} is a tree, we see that Q_1 is also a tree query.

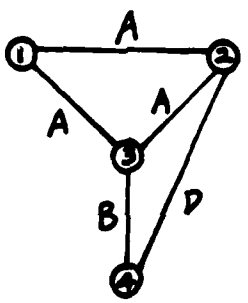
2. $Q_3=(q_3, TL)$, where $q_3 = (R_1.A = R_2.A) \wedge (R_2.A = R_3.A) \wedge (R_3.B = R_4.B) \wedge (R_2.D = R_4.D)$

The qual graph G_{Q_3} of Q_3 is not a tree.

G_{Q_3}



Its transitive closure $G^+_{Q_3}$ is



From $G^+_{Q_3}$ we can see that all queries equivalent to Q_3 have cyclic qual graphs. Thus, Q_3 is a cyclic query.

Note that, for this thesis, we will consider an operation or a query to be a mapping from a temporary database state to a new temporary database state. During the analysis of a query solution strategy, the database state is

conceptually considered changing from one state to the other state by an operation. The real database state stored in the system does not change unless update operations are really performed.

2.3 Review of Previous Works

2.3.1. [WONG 77] & [ESW 78]

Wong's algorithm is the first comprehensive solution to the distributed query processing problem (DQPP). It was implemented in SDD-1. The assumption of the system environment is the following:

1. Each system is a relational, nonredundant DBMS (i.e. unique copy of data).
2. The final result of a query is produced at a single site.
3. Each system can MOVE fragments of relations to another system.
4. The communication cost is a function of transmitted data volume and the goal is to minimize the communication cost.

This algorithm translates a query Q into a sequence of relational algebra operations (selection, projection and join) and MOVE operations (move portions of relations from one site to another). It first selects a final processing site, S_0 , and constructs an initial feasible solution:

Move all R referenced by Q to S_o.

Process Q at S_o as a local query.

The solution is improved by recursively replacing a MOVE by lower cost sequences of MOVEs and relational operations. It terminates when no MOVE can be replaced by a lower cost sequence. This algorithm produces increasingly efficient sequences of commands by its hill-climbing discipline. Since at each step of refinement the best alternative is chosen, this algorithm can be thought as a greedy heuristic algorithm. The critical point of this algorithm is that the heuristic is too weak to guarantee optimality and it has no analytical tool for the evaluation of traffic volumes when MOVE is executed.

Wong's algorithm has been also adopted for the distributed version of INGRES [ESW 78]. The algorithm begins by executing all one variable subqueries to obtain reduced relations. Each site sends a short description of these relations to the MASTER INGRES site where the query has been originated so that the MASTER site knows which sites are involved in the query processing. This algorithm then breaks the qualification into separate pieces using a few simple heuristics. Consequently, the sequence of distributed operations is decided by means of the quantitative information obtained. Two cost criteria, minimum response time and minimum communication traffic are considered.

2.3.2. [HY 79]

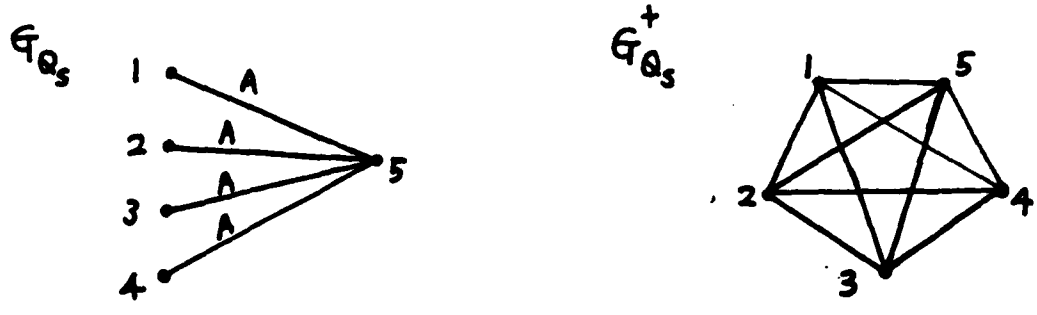
Hevner and Yao present another approach to distributed query processing that can be interpreted as a semijoin approach. They consider a class of simple queries in which, after initial local processing, each relation in the query contains only one attribute --- the common joining attribute. All relations are joined on this single attribute. If we assume the only attribute name is A, the qualification of a simple query can be written as

$$q_S = \bigwedge_{i=1}^{n-1} (R_i.A = R_n.A)$$

By the transitive rule, the closure is

$$q_S^+ = \bigwedge_{i=1}^{n-1} \bigwedge_{j \neq i} (R_i.A = R_j.A)$$

The qual graphs of q and q^+ (for n=5) are



From the qual graph $G^+_{Q_S}$, It is easy to see that any spanning tree of $G^+_{Q_S}$ corresponds to a query equivalent to

the simple query.

Note that for a simple query, intersection, join and semijoin are identical operations. The optimization problem for this class of queries is trivial. They propose two algorithms whose cost functions are the global response time and response time related only to the network traffic (total transmission time). The first one implies the maximum degree of parallelism of query execution. The quadratic time bound is proved for this case. The second strategy corresponds to the minimization of network traffic and a linear time algorithm is presented.

The authors attempt to generalize this algorithm for arbitrary equi-join queries. For a relation $R_i = (S_i, r_i)$, where $S_i = \{A_{i1}, A_{i2}, \ldots, A_{iK_i}\}$, they define the selectivity for each domain $Dom(R_i.A_{ij})$ to be the number of values of $Dom(R_i.A_{ij})$ currently appearing in the column $R_i.A_{ij}$ of relation state r divided by the cardinality of $Dom(R_i.A_{ij})$. They assumed the selectivity on one domain does not affect the selectivity of the other joining domain. Therefore, each joining domain in the relation R_i is handled separately. A heuristic algorithm that uses an improved exhaustive search is proposed for the general queries.

2.3.3 [GBWRR 81], [BG 81] and [BG 80]

In [GBWRR 81], an algorithm is proposed in which semi-join concepts are exploited in order to refine the

approach of Wong's algorithm. In this algorithm, a query Q is processed in two phases. Phase 1 is called the reduction phase. It seeks to reduce the database state with respect to Q. The use of semijoins is the principal tactic in this phase. A sequence of semijoin operations SJ is said to reduce a database state D for query Q if we may apply SJ to D without affecting Q's answer, i.e. Q(D)=Q(SJ(D)). A sequence of semijoins is called a reducer for Q if it reduces every database state for Q. A sequence of semijoins SJ is cost effective in state D if the amount of data requiring inter-site data transmission in order to compute SJ(D) is less than or equal to the quantity of data in D that will be eliminated by SJ. The goal of the reduction phase is to translate Q into a cost effective reducer. This reduction phase is known as the full reducer problem.

Phase 2 is the final processing phase. The system selects one site as the final processing site and the reduced databases of the other sites are transmitted to the final site. The system then excutes Q against these databases at that site as a local query. The final processing phase of SDD-1 is very simple. The core of the SDD-1 query processing algorithm is the reduction phase.

The authors present a heuristic algorithm that solves this problem for a class of equi-join queries. By defining rules for estimating the cost and effectiveness of semijoins, the algorithm starts with an initial feasible

solution consisting of null reducer, i.e. an empty sequnce of semijoins. The algorithm improves the initial solution by iteratively appending cost effective semi-joins to it. When all cost effective semi-joins have been exhausted, the basic optimization is complete. At this point the algorithm is the same as Wong's algorithm. Let the sequence of semijoins just constucted be SJ. The algorithm next permutes the order of semijoins in SJ such that the effectiveness of SJ is increased while its cost is decreased. Finally, the algorithm selects a final processing site and prunes semijoins in SJ that are made unnecessary by the choice of final site. The resulting sequence of semijoins is executed and quaranteed to be a cost effective reducer for Q. It is still not guaranteed to be optimal.

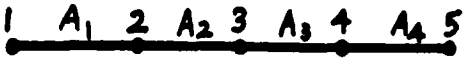
2.3.4 [CHIU 79] and [CH 80]

The use of semijoins for distributed query processing is also studied by Chiu. He considers a sub-class of equi-join queries whose qualification can be written as

$$q_C = \bigwedge_{i=1}^{n-1} (R_i.A_i = R_{i+1}.A_i) \qquad A_i \neq A_j$$

The qual graph of this query when n=5 is:

$G^+_{Q_c} = G_{Q_c}$



In this class of query, every joining attribute of one relation joins to exactly one attribute of another relation. This sub-class is called a chain query because the graph representation of it is a chain. This class of queries has the property that the closure of the qual graph G is the same as G .

They proved that semijoins are powerful enough to solve chain queries. They are the only authors who present a syntatic characterization of chain queries and derive an efficient dynamic programming algorithm that translates any chain query Q into an optimal sequence of semijoins that compute Q. This algorithm has $O(n^3)$ time complexity, where n is the number of relations referenced by Q. They generalize this approach to a larger class of queries called tree queries whose answer is a subset of one relation in the database, and develop a methodology for optimally solving this class of tree queries. However, the syntatic charaterization becomes more complicated. No timing analysis is presented for the tree query case.

2.3.5 Summary

Most of the distributed query processing algorithms developed to date have the following common features:

1. single query processing

Most of the algorithms only consider the optimization of the processing of one query, as if DBMSs were a single

user system. In fact, every node in DBMSs is simultaneously an access point of DBMSs. Moreover, every node has the ability to do query decomposition and to process relational operators. It is desired to have an optimal processing algorithm for the set of queries which are in the system at a given time interval.

2. communication cost dominance

The inter-system communication time is the dominant cost of distributed query processing. The communication time is dependent on the volume of data to be transmitted. The total transmission time of a query is proportional to the amount of data and messages required to be transmitted. Most of the algorithms consider only the transmission cost. It would be desirable to develop algorithms that consider local processing costs as well. Another direction would be to select a better execution sequence for semijoin and join operations and exploit the feature of parallelism of data transmissions over links and local processings.

3. heuristic algorithm

For each query processing strategy, the costs of one step in the execution depend on previous steps. The set of strategy space blows up very quickly as the number of steps increase. This suggests that the distributed query processing problem may inherently be a complex problem. Most of the distributed query processing algorithms are heuristics. [CHIU 79] is the only one who studies the

syntatic characteristics of semijoin programs for a class of chain queries in order to reduce the set of strategy space. It is our desire to study the complexity of the distributed query processing problem. We are also interested in the understanding of quantitative characteristics of this problem.

2.4 A Model for Equi-join Query Processing

As we reviewed in the last section, most of the distributed query processing algorithms proposed have the common philosophy of performing local processing first, then applying as many semijoins as possible to reduce the database state as much as possible and then sending to the final site to perform the join and produce the final results. The reason for doing so is presumably that the semijoin tactic will be profitable.

As described in [ULL 80], there are some rules which may be used to help "optimize" relational expressions, although these rules in no sense guarantee optimal overall equivalent expressions. The basic idea is to attempt to perform selections and projections as early as possible.

For a query Q=(q,TL), let $\{ R_1, R_2, \ldots, R_n \}$ be the set of relation schemas referenced by q and let X be the set of attributes appearing in q. Before processing the query, we can project each relation R_i over attributes $(X \cup TL) \cap R_i$. We then execute those subqueries which reference only one

local relation. We may also want a cascade of those operations to be organized into one selection followed by one projection and group selections and projections with the preceding binary operation. From here on, we represent $R_1, R_2, \ldots, R_n$ as the relations after such preprocessing.

In a distributed query processing environment, if we adopt the assumption that the data communication cost dominates the local processing cost, then the local processing costs of a query (e.g. select, project) are negligible. The only significant cost needed to be considered is the data transmission cost. Data transmission is incurred when two relations that must be joined may reside at different sites. To perform the join, one way is to move the entire relation from one site to the other. The other way is to replace a join by performing semijoins first and then performing join. Assume R_1 and R_2 at different sites and we want to join R_1 and R_2 at the site of R_2. By the semijoin strategy, one can send the projection of R_2 on its joining columns to R_1's site and perform a semijoin to reduce R_1 by R_2 before sending R_1 to R_2's site. This will be a profitable tactic only when the projection of R_2 on its joining columns is smaller than the amount by which R_1 is reduced by the semijoin. From the above example of joining two relations, one can easily be convinced that semijoins-then-joins stategies may not be able to produce an optimal strategy for the objectives of the minimization of

the total data transmission cost. Our approach is to extend the strategy space to the class of joins-semijoins-mixed strategies. A joins-semijoins-mixed strategy is an ordered sequence of join and semijoin operations where join and semijoin operations intermingle with each other. (i.e. The order of join and semijoin operations do not have any restriction.)

We assume that a query Q, specified by a qualification q over the relations R_1, R_2,...,R_n, and by a target list TL, can be decomposed into a set of operations $\{ p_1, p_2, \ldots p_k \}$ which will produce the answer to the query, where $p_k \in \mathcal{A}$, the set of relational algebra operators. In general, a query can be decomposed into several different executing sequences which will produce the same answer. We call such an executing sequence a strategy. Let S(Q) denote the set of strategies which answer the query Q. The goal of the problem is to minimize the overall cost of executing this query Q. We can formulate this problem as

$$\underset{P \in S(Q)}{\text{MIN}} \; f(P, D[0]) = \sum_{i=1}^{l} f_i(p_i, D[i])$$

$$\text{s.t. } P = p_1 p_2 \ldots\ldots p_l$$

$$D[i+1] = p_i(D[i])$$

D[0] is the initial database state

Here $p_i(D[i])$ means the mapping from the temporary database state at stage i by the operation p_i to a new temporary database state. During the analysis of a query solution strategy, the database state is conceptually considered changing from one state to the other state by an operation. The real database state stored in the system does not change unless update operations are performed.

As shown in [BG 81], any query Q=(q,TL) with an equijoin qualification q and a target list TL can be efficiently transformed by renaming attributes of the relation schema and qualification into an equivalent natural join query. Instead of the class of equijoin queries, EQJ, we shall study the class of natural join queries, NJQ.

In this section, we restrict our study to a class of queries that after initial local processing and attribute renaming, the resulting queries are natural join queries. Although it is a subset of the complete relational calculus language, it is a rich and large class of queries in practice.

2.4.1 Definitions and Assumptions

We assume a distributed database management system DDBMS consists of a collection of interconnected computers S_1, S_2, ..., S_n at different sites. Each computer, known as a node in the network, contains a DBMS. Data are logically viewed in the relational model. Without loss of generality, we assume each site only consists of one relation. In the distributed database DD={ D_1, D_2, ..., D_n }, where each D_i only consists of one relation R_i, we shall use DD={ D_1, D_2, ..., D_n } or { R_1, R_2, ..., R_n } interchangeably when no confusion will occur.

Data transmission in the network is via communication links. We assume that the transmission cost to send one byte of data between any two sites i & j is known and equal to c_{ij}. Thus the cost function of transmitting data of volume V between two sites i & j is a linear function $C_{ij}(V)=c_{ij}*V$. We assume that all possible subqueries involving data at a single site are preprocessed; This we call "local processing". The effect of local processing is to reduce the amount of data that needs further processing. We will regard the state of each database as the resulting state of the database after local processing. Thus, after local processing, the following parameters of the qeury can be defined.

n = number of sites (i.e. relations) in the remaining query

a_i = | R_i | , number of attributes in site R_i

Y_{ij} = $R_i \wedge R_j$, the set of attributes of joining domains between R_i & R_j

v_i = number of tuples in relation R_i

w(A) = the width of data item of attribute A

$s_i = v_i * \sum_{A \in R_i} w(A)$, the size of the relation R_i

$w(Y_{ij}) = \sum_{A \in Y_{ij}} w(A)$

In DDBMS, we define two types of directed operators.

Definition:

1. $<|X|_{ij}$ (or $R_i <|X| R_j$) is the directed natural join operator which sends R_j to R_i and performs the natural join of R_i and R_j at R_i's site.

2. $<|X_{ij}$ (or $R_i <|X R_j$) is the directed natural semijoin operator which projects $Y_{ij} = R_i \wedge R_j$ over R_j , sends the result to R_i and performs the join of R_i and that result at R_i's site. (i.e. R_i $|X| \pi_{Y_{ij}} R$ at R_i 's site).

Note that $|X|>_{ij} = R_i |X|> R_j$ and $X|>_{ij} = R_j X|> R_j$ are similarly defined. One can use them interchangeablly. The semijoin operation only reduces the relation state without changing the relation schema.

Definition:

A join-semijoin program $P=p_1 p_2 \ldots p_\ell$ is a sequence of directed natural join and directed natural semijoin operators.

A natural join qualification q with final node at R_1 can be done by sending all relations R_i, $i\neq 1$, to R_1 and performing $R_1 |X| R_2 |X| \ldots |X| R_n$ at node R_1. So $R_2 |X| > R_1$, $R_3 |X| > R_1$, ... $R_n |X| > R_1$ or its permutation are join-semijoin programs of this qualification q.

2.4.2 Query Processing Graph

We define a processing graph of a qualification over a database schema $DD=\{R_i\}_{i=1}^{n}$ to be a graph with two types of edges, $<V_1, A_1, B_1>$. V_1 is the set of nodes, which is equal to D. A_1 is a set of semijoin edges which is $\{a_{ij} \mid R_i \cap R_j \neq \emptyset$ and $R_i \not\subseteq R_j\}$. We denote such an edge by i -->-- j with one arrow on the edge. $B_1 = V_1 \times V_1 = \{b_{ij} \mid i \neq j\}$ is the set of join edges. We denote such an edge by i -->>-- j with two arrows on the edge.

Note that if $R_i \cap R_j = \emptyset$, then we can not perform a semijoin between R_i and R_j, so a_{ij} is not a semijoin edge. If $R_i \subseteq R_j$, then $R_i = R_i \cap R_j$. The semijoin of R_i to R_j, $R_i X| > R_j$, is the same as the join of R_i to R_j, $R_i |X| R_j$. This operation is covered by join edge b_{ij}.

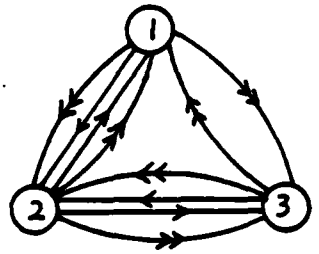
Example:

$R_1=\{ A_1, A_2, A_3, A_4 \}$

$R_2=\{ A_3, A_4, A_5, A_6 \}$

$R_3=\{ A_5, A_7, A_9 \}$

The processing graph of the natural join qualification q is:



Without lost of generality, from now on we assume that the final node of a query is node 1.

Definition:

Given a natural join qualification q ,a join-semijoin program P is said to be correct with respect to q if after executing the program P, the final node will have a new relation $R_1' = R_1 |X| R_2 |X| \ldots |X| R_n$.

Lemma 1:

A join-semijoin program consisting of a directed path of edges in B_q from R_{k_0} to R_{k_l}, $b_{k_0k_1}, b_{k_1k_2} \ldots, b_{k_{l-1}k_l}$ will form a relation $R_{k_0} |X| R_{k_1} |X| \ldots |X| R_{k_l}$ in node R_{k_l} .

Proof: We prove this by induction on the length of the path. If l=1, then the path is $b_{k_0k_1}$. After this

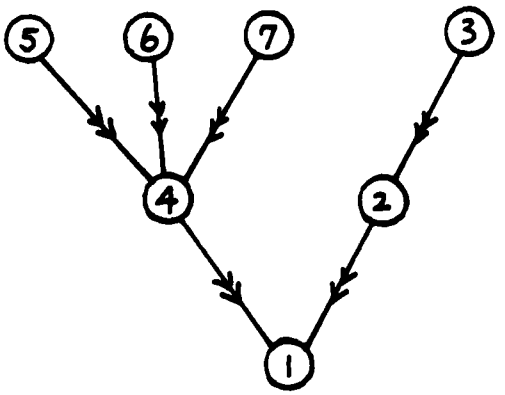
operation we will have $R'_{K_{\ell-1}} \leftarrow\!\!- R_{K_0}|X|R_{K_1}$. By the induction assumption on 1-1, $R'_{K_{\ell-1}} \leftarrow\!\!- R_{K_0}|X|R_{K_1}|X|\ldots|X|R_{K_{\ell-1}}$. In the case of 1, for the first 1-1 edges of this path, $R'_{K_{\ell-1}} = R_{K_0}|X|R_{K_1}|X|\ldots|X|R_{K_{\ell-1}}$ by induction assumption. After performing $b_{K_{\ell-1}K_\ell}$, we will have $R'_{K_\ell} = R'_{K_{\ell-1}}|X|R_{K_\ell} = R_{K_0}|X|R_{K_1}|X|\ldots|X|R_{K_\ell}$.

Definition:

Given a directed spanning tree T toward final node R_1 , a program of operations in B_2 , $b_{K_1K_2}, b_{K_3K_4}, \ldots, b_{K_{\ell-1}K_\ell}$ is said to associate with T if each directed path in the directed spanning tree has the same ordering as the subsequence of corresponding operations in the program.

Example: In the following directed spanning tree T_0 toward node 1, $b_{54}\, b_{64}\, b_{74}\, b_{41}\, b_{32}\, b_{21}$ is a program associated with T_0 and $b_{32}\, b_{74}\, b_{64}\, b_{54}\, b_{21}\, b_{41}$ is another program associated with T_0 .

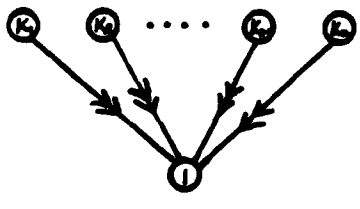
T_0

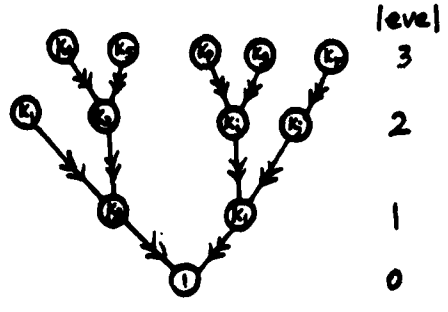


Lemma 2:

A program with edges in B_9 associated with a directed spanning tree with nodes k_1 , k_2 ,...,k_n and toward node 1 will form the joining of R_{K_1} ,R_{K_2} ,...,R_{K_n} ,R_1 (i.e. $R_{K_1}|X|R_{K_2}|X|...|X|R_{K_n}|X|R_1$) at node 1.

Proof: we prove this by induction on the height of the spanning tree. If h=2, then the resulting relation at node 1 is the joining of R with all leaf relations which is $R_{K_1}|X|R_{K_2}|X|...|X|R_{K_n}|X|R_1$ at node 1 .(see Figure) By the induction assumption on h-1, the resulting relation at the final node 1 is the joining of all the relations in the nodes of the tree. In the case of h, each node at level 1 has height h-1, and the resulting relation at these level 1 nodes is the joining of all the relations in the nodes of the corresponding subtree above that node. Consider the final node 1; it will join all the resulting relations in the level 1 nodes of node 1 with the relation R_1 . Because a directed spanning tree will contain each node k exactly once, we will obtain the relation $R_{K_1}|X|R_{K_2}|X|...|X|R_{K_n}|X|R_1$ at node 1 .





Theorem 1:

Let Q=(q,TL) be a natural join query and TL=R_1 U...U R_n . Let P be a join-semijoin program for q; then P is correct for q iff there exists an ordered subset of the set $\{b_{ij}\}$ in P which associates with a directed spanning tree toward node R_1.

Proof: IF: Since a natural semijoin from relation R to S only reduces relation state s to a new state consisting of tuples with values in the columns of joining attributes appearing in both r and s, it does not change the relation schema S. Thus after performing the sequence of join opeations associated with the directed spanning tree toward R_1, we get R_1 |X|...|X|R_n at node R_1. Any other join operation does not change the state. This implies P is correct for q.

ONLY IF: Let P be a correct program for P. For each semijoin operation a_{ij} in P, $R_i \cap R_j \neq \emptyset$ and $R_i \not\subseteq R_j$. Thus, performing the semijoin operation does not move the full relation state from node R_i to node R_j . We still need to perform a join to move the full table of R_i to R_j. If there does not exist a subset of $\{b_{ij}\}$ in P which form a directed spanning tree toward node R_1, then there is some node R_k which is disconnected from the tree component. Then some information from those nodes which do not have a

path toward R_1 is lost. So, in order to form $R_1 |X| R_2 |X| \ldots |X| R_{n-1} |X| R_n$ at node R_1, there must be a subset of edges in B_q that forms a directed spanning tree toward node R_1.

From theorem 1, we know that given a NJQ qualification q, the set of correct programs for q is the set of join-semijoin programs such that there exists a directed spanning tree toward R out of the set of join edges in P. We denote this set of correct programs for q by $\mathcal{P}(q)$. In this thesis, we restrict the problem by only looking for a best program within this class of programs. The distributed query processing problem becomes to find a program $P \in \mathcal{P}(q)$ with minimum communication cost. For a program p, if we change the order of the sequence of operations, the total communication cost will be different. The set of correct programs $\mathcal{P}(q)$ is very large. In fact, after executing one operation in P, the number of rows and columns of some relations will be changed. This change then affects the communication cost of the next operation. So the communication cost of one operation will depend on the previous subsequence of operations.

2.4.3 Estimate the Size of the Derived Relations

In order to compare the communication cost of query processing strategies, it is very important to have a method of estimating the size of a relation after one operation.

Also the system for estimation of the size of the derived relation must be consistent in the sense that if two sequences of operations will produce the same results, the estimated sizes of the result according to the two sequences of operations must be the same. In this section, we use capital R to represent both relation state and relation schema.

We introduce the notion of semijoin reducibility and join reducibility of R_i to R_j, denoted by α_{ij} and β_{ij} respectively, for each pair of relations R_i and R_j, where $0 \leq \alpha_{ij} \leq 1$ and $0 \leq \beta_{ij} \leq 1$. The interpretation of the semijoin reducibility α_{ij} of R_i to R_j is the percentage of rows of R_j that are eliminated after performing the semijoin R_i X|> R_j. At stage t, if the number of rows of R_j is $v_j[t-1]$ and the semijoin reducibility of R_i to R_j is $\alpha_{ij}[t-1]$, then the number of rows of R after performing semijoin R_i X|> R_j will be reduced to $v_j[t] = v_j[t-1] * (1-\alpha_{ij}[t-1])$. Note that the semijoin reducibility of R_i to R_j is not necessarily equal to the semijoin reducibility of R_j to R_i and $\alpha_{ii}[t]=0$ for all t. The interpretation of the join reducibility of R_i to R_j is that after performing join R_i |X|> R_j, the number of rows of the new relation $R_i|X|R_j$ at site j will be $v_j[t] = v_i[t-1] * v_j[t-1] * (1-\alpha_{ij}[t-1]) * (1-\alpha_{ji}[t-1]) * (1-\beta_{ij}[t-1])$. This is because the effect of join R_i |X|> R_j is equivalent to performing the semijoins R_iX|> R_j and R_jX|> R_i and then performing the join of R_i to R_j. Both semijoin

reduciblities and join reducibility affect the number of rows of the new relation. The join reducibility of R_i to R_j is the same as the join reducibility of R_j to R_i . i.e. $\beta_{ij}[t] = \beta_{ji}[t]$. Also $\beta_{ii}[t]=0$ for all t.

For this paper, we assume that the number of tuples of relation R_i in the system, v_i , and the set of reducibilities $\{\alpha_{ij}, \beta_{ij}\}$ of each pair of relations are known. Note that these quantities depend on the initial local processing and attribute renaming process of any given query. After the systems are running, they can be updated periodically according to statistical measurements. They will form the basic information for processing a given query.

Since the number of rows and columns of a relation will be changed after one operation, the reducibilities of this relation with other relations will be changed too. We define how the reducibilities will be changed after one operation. Assume the databases before the operation p_t to be $D=\{R_1[t-1],...R_n[t-1]\}$, the number of rows of each relation $R_i[t-1]$ to be $v_i[t-1]$, and the semijoin and join reducibilities of $R_i[t-1]$ to $R_j[t-1]$ to be $\alpha_{ij}[t-1]$ and $\beta_{ij}[t-1]$.

If the operation p_t at stage t is a_{ij} , i.e. $R_i\ X|{>}R_j$, then the database schema will remain the same. The state at node j will change to $R_j[t]=R_i[t-1]\ X|\ R_j[t-1]$ and all

other states will remain the same, i.e. $R_K[t]=R_K[t-1]$, $\forall$ k $\neq$ j. The number of rows of relation $R_j[t]$ will be changed to equal $v_j[t-1] * (1-\alpha_{ij}[t-1])$ and the number of rows of all other relations will remain the same. At stage t, we have

According to the definition of reducibilities, the estimated size of R_i X|(R_i X|R_j) is

$v_j[t]*(1-\alpha_{ij}[t]) = v_j[t-1]*(1-\alpha_{ij}[t-1]) * (1-\alpha_{ji}[t])$.

Since R_i X|(R_i X|R_j) =R_i X|R_j ,

$v_j[t-1] *(1-\alpha_{ij}[t-1]) * (1-\alpha_{ij}[t])$

$=v_j[t-1] * (1-\alpha_{ij}[t-1])$,

which implies $\alpha_{ij}[t]=0$.

The estimated size of (R_i X|R_j) X|R_i is

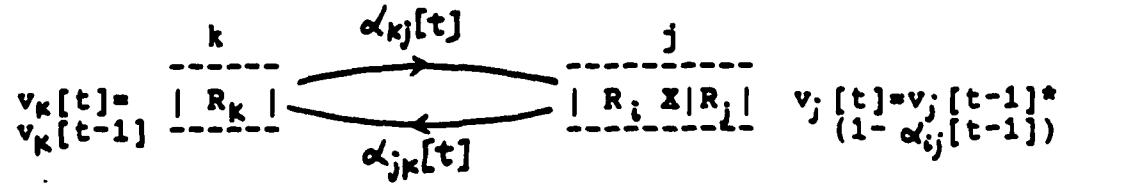
$v_i[t] * (1-\alpha_{ji}[t]) = v_i[t-1] * (1-\alpha_{ji}[t])$.

Since (R_i X|R_j) X| R_i = (R_j X|R_i),

$v_i[t-1] *(1-\alpha_{ji}[t]) = v_i[t-1] * (1-\alpha_{ji}[t-1])$,

which implies $\alpha_{ji}[t]=\alpha_{ji}[t-1]$.

Next, we consider the reducibilities $\alpha_{Kj}[t]$ and $\alpha_{jK}[t]$. At stage t, we have



Lemma: $R_k X|(R_i X|R_j) = R_i X|(R_k X|R_j)$

Proof:

$$R_k X|(R_i X|R_j)$$
$$= \{t \in R_j \mid t \in R_i X|R_j \ \& \ \exists\, s_k \in R_k \text{ , s. t. } t[Y_{jk}] = s_k[Y_{jk}]\}$$
$$= \{t \in R_j \mid \exists\, s_i \in R_i \ \& \ \exists\, s_k \in R_k \text{ , s. t. } t[Y_{ij}] = s_i[Y_{ij}] \ \& \ t[Y_{kj}] = s_k[Y_{kj}]\} \quad (*)$$

Similarily, we have $R_i X|(R_k X|R_j) = (*)$.

Thus, $R_k X|(R_i X|R_j) = R_i X|(R_k X|R_j)$.

The size of $R_k X|(R_i X|R_j)$ is

$$v_j[t-1]*(1-\alpha_{ij}[t-1])*(1-\alpha_{kj}[t])$$

and the size of $R_i X|(R_k X|R_j)$ is

$$v_j[t-1]*(1-\alpha_{kj}[t-1])*(1-\alpha_{ij}[t]).$$

By the above lemma, they should be equal.

In extreme case, $\alpha_{kj}[t]$ could be either 0 or 1. In general, we will find an approximation function of $\alpha_{kj}[t]$ Let

$J(Y_{jk})$ =the set of values in columns Y_{jk} of R_j .

$K(Y_{jk})$ =the set of values in columns Y_{jk} of R_k .

$J_i(Y_{jk})$=the set of values in columns Y_{jk} of $R_i X|R_j$.

Here we assume the set of elements in $J(Y_{jk})$ is uniformly reduced by the operation a_{ij} , i.e. the percentage of elements being reduced in the set of common elements, $J(Y_{jk}) \cap K(Y_{jk})$ and in the remaining set of non-common elements, $J(Y_{jk}) - K(Y_{jk})$ in $J(Y_{jk})$ by operation a_{ij} are the same. Thus, the

ratio of the size of the set, $J_{\hat{\alpha}}(Y_{jK})$ $K(Y_{jK})$, of common elements in $J_{\hat{\alpha}}(Y_{jK})$ with the size of the set, $J_{\hat{\alpha}}(Y_{jK})$, in $J_{\hat{\alpha}}(Y_{jK})$ is the same as the ratio of the size of the set, $J(Y_{jK}) \cap K(Y_{jK})$, of common elements in $J(Y_{jK})$ with the size of the set, $J(Y_{jK})$, in $J(Y_{jK})$ before operation a_{ij}, i.e.

$$\frac{|J_{\hat{\alpha}}(Y_{jK}) \cap K(Y_{jK})|}{|J_{\hat{\alpha}}(Y_{jK})|} = \frac{|J(Y_{jK}) \cap K(Y_{jK})|}{|J(Y_{jK})|}$$

So we have

$$\alpha_{Kj}[t] = \alpha_{Kj}[t-1].$$

This implies the size of R_K $X|(R_{\hat{\alpha}}$ $X|R_j)$ is

$$v_j[t-1]*(1-\alpha_{ij}[t-1])*(1-\alpha_{Kj}[t-1]).$$

Following the same assumption as above, after operation a_{ij}, the number of common elements in columns Y_{jK} of R_i $X|R_j$ and R_K has been reduced by $(1-\alpha_{ij}[t-1])$. After performing semijoin operation $(R_i$ $X|R_j)X|>R_K$, the size of $(R_i$ $X|R_j)X|R_K$ will be

$$v_K[t-1]*(1-\alpha_{ij}[t-1])*(1-\alpha_{jK}[t-1]).$$

By the definition of semijoin reducibility, the size of $(R_i$ $X|R_j)X|R_K$ will be

$$v_K[t-1]*(1-\alpha_{jK}[t])$$

where $\alpha_{jK}[t]$ is the semijoin reducibility of $R_i X|R_j$ to R_K. This implies

$$1-\alpha_{jK}[t] = (1-\alpha_{ij}[t-1])*(1-\alpha_{jK}[t-1]).$$

Thus, $\alpha_{jK}[t] = \alpha_{jK}[t-1] + \alpha_{ij}[t-1] - \alpha_{jK}[t-1]*\alpha_{ij}[t-1]$.

for $k \neq i,j$.

For the join reducibility $\beta_{jk}[t]$ of $R_i\ X|R_j$ with R_k, if we want to still have $\beta_{jk}[t]$ defined by pairwise formula

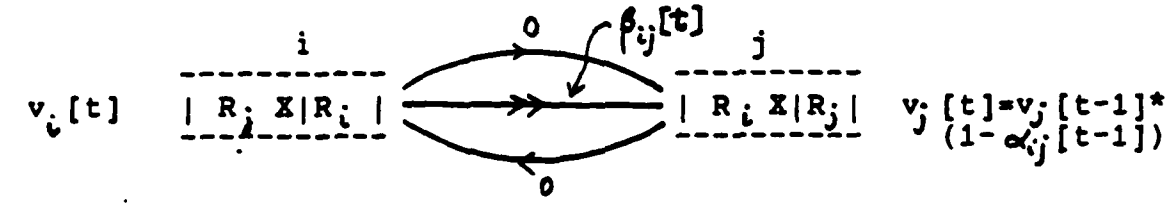
$$v_j[t]^* v_k[t]^*(1-\alpha_{jk}[t])^*(1-\alpha_{kj}[t])^*(1-\beta_{jk}[t]),$$

then we must have

$$1-\beta_{jk}[t] = (1-\beta_{jk}[t-1])/(1-\alpha_{ij}[t-1]).$$

We will discuss the reasons later. Here we assume $\beta_{jk}[t] = \max\{1-(1-\beta_{jk}[t-1])/(1-\alpha_{ij}[t-1]), 0\}$.

Next, we consider reducibility $\beta_{ij}[t]$. Suppose at stage t, we have $R_j\ X|R_i$ at node i and $R_i\ X|R_j$ at node j.



then $\alpha_{ij}[t]=0$, $\alpha_{ji}[t]=0$, $v_i[t]=v_i[t-1]^*(1-\alpha_{ji}[t-1])$ and $v_j[t]=v_j[t-1]^*(1-\alpha_{ij}[t-1])$.

Since $(R_j\ X|R_i)\ |X|\ (R_i\ X|R_j) = R_i\ |X|R_j$, it follows that

$$v_i[t-1]^*(1-\alpha_{ji}[t-1])^* v_j[t-1]^*(1-\alpha_{ij}[t-1])^*(1-\beta_{ij}[t-1])$$
$$= v_i[t-1]^*(1-\alpha_{ji}[t-1])^* v_j[t-1]^*(1-\alpha_{ij}[t-1])^*(1-\beta_{ij}[t]).$$

Thus $\beta_{ij}[t]=\beta_{ij}[t-1]$.

Figure 2.1 illustrate these changing rules. We summarize the reducibilities changing rules after semijoin operation a in the following:

$$\alpha_{ij}[t]=0$$
$$\alpha_{jk}[t]=\alpha_{jk}[t-1]+\alpha_{ij}[t-1]-\alpha_{jk}[t-1]^*\alpha_{ij}[t-1] \quad \forall\ k\neq i,j.$$
$$\alpha_{pq}[t]=\alpha_{pq}[t-1] \qquad \text{otherwise.}$$

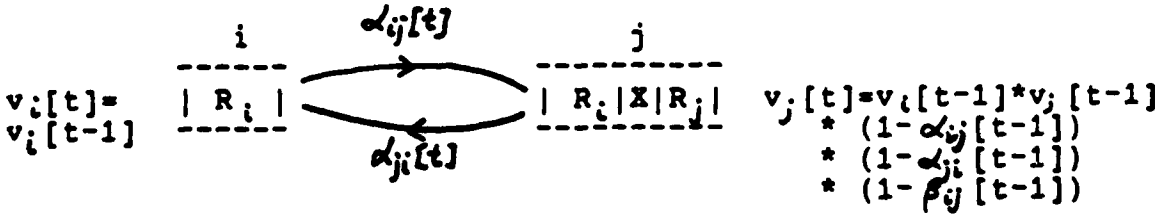
$\beta_{jk}[t] = \max\{ 1-(1- \beta_{jk}[t-1])/(1- \alpha_{ij}[t-1]), 0\}$. $\forall k \neq i,j$.

$\beta_{pq}[t] = \beta_{pq}[t-1]$ otherwise.

If the operation p_t at stage t is b_{ij}, i.e. $R_i |X| > R_j$, then the database schema at node j, $R_j[t]$ will change to $R_i[t-1] \cup R_j[t-1]$, and the relation state at site j will be $R_i[t-1] |X| R_j[t-1]$. The number of rows of $R_j[t]$, $v_j[t]$, is $v_j[t-1] * v_i[t-1] * (1- \alpha_{ij}[t-1]) * (1- \alpha_{ji}[t-1]) * (1- \beta_{ij}[t-1])$.

All other relation states will remain the same. Because this is a join operation, the semijoin reducibilities and join reducibilities will be affected. (See Figure 2.2)

At stage t, we have



According to the definition of reducibilities, the estimated size of $R_i |X| (R_i |X| R_j)$ is

$$v_j[t] * (1- \alpha_{ij}[t]).$$

Since $R_i |X| (R_i |X| R_j) = R_i |X| R_j$, this implies

$$1- \alpha_{ij}[t] = 1, \text{ i.e. } \alpha_{ij}[t] = 0.$$

Similarily, the estimated size of $(R_i |X| R_j) |X| R_i$ is

$$v_i[t] * (1- \alpha_{ji}[t]) = v_i[t-1] * (1- \alpha_{ji}[t]).$$

Since $(R_i\ |X|R_j)\ X|R_i = R_j\ X|R_i$,

$$v_i[t-1] * (1-\alpha_{ji}[t]) = v_i[t-1] * (1-\alpha_{ji}[t-1])$$

which implies $\alpha_{ji}[t] = \alpha_{ji}[t-1]$.

Lemma: $(R_i\ |X|R_j)\ \ X|\ R_k \subseteq R_i\ X|\ (R_j\ X|\ R_k)$

with equality when $Y_{ij} \subseteq Y_{ik} \cap Y_{jk}$.

Proof: Let $Y_{ijk} = Y_{ij} \cap Y_{jk} \cap Y_{ik}$. Then

$$(R_i\ |X|R_j)\ \ X|\ R_k$$

$$= \{t \in R_k \mid \exists\ s_0 \in R_i|X|R_j \ni s_0[Y_{ik} \cup Y_{jk}] = t[Y_{ik} \cup Y_{jk}]\}$$

$$= \{t \in R_k \mid \exists\ s_1 \in R_i\ \&\ s_2 \in R_j\ \text{ s. t.}$$

$$t[Y_{ik}] = s_1[Y_{ik}]\ \&$$

$$t[Y_{jk}] = s_2[Y_{jk}]\ \&$$

$$s_1[Y_{ij}] = s_2[Y_{ij}]\ \}.$$

Also

$$R_i\ X|\ (R_j\ X|\ R_k)$$

$$= \{t \in R_k \mid \exists\ s_3 \in R_j \ni t[Y_{jk}] = s_3[Y_{jk}]\ \&$$

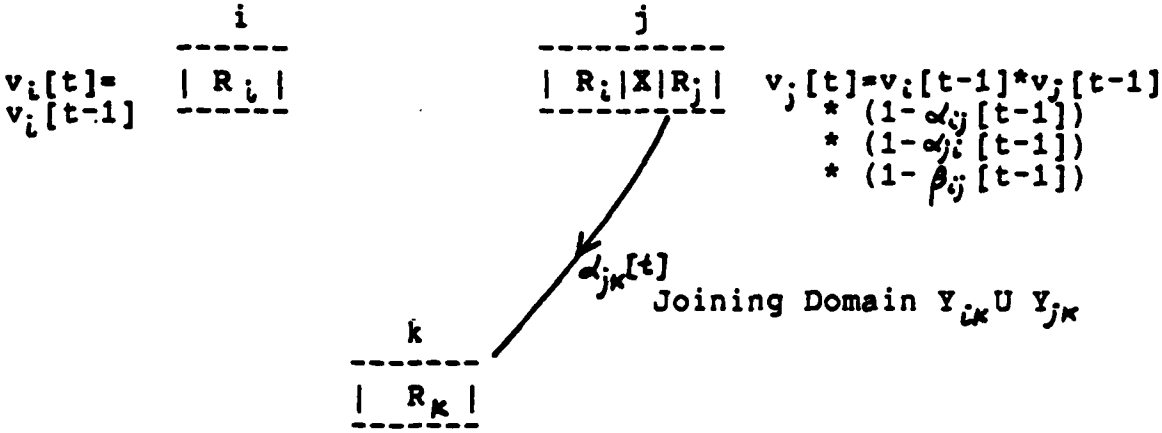
$$s_4 \in R_i \ni t[Y_{ik}] = s_4[Y_{ik}]\ \}.$$

Thus $(R_i\ |X|R_j)\ \ X|\ R_k \subseteq R_i\ X|\ (R_j\ X|\ R_k)$.

If $Y_{ij} \subseteq Y_{ik} \cap Y_{jk}$, then $s_1[Y_{ij}] = t[Y_{ij}] = s_2[Y_{ij}]$.

This implies equality of relation states when $Y_{ij} \subseteq Y_{ik} \cap Y_{jk}$.

Because of the above lemma, we approximate the number of rows of $(R_i\ |X|R_j)\ \ X|\ R_k$ by the number of rows of $R_i\ X|\ (R_j\ X|\ R_k)$. If at stage t, we have



the joining domain of R_i |X| R_j with R_k will be $Y_{ik} U Y_{jk}$.

Since $v_k[t] * (1-\alpha_{jk}[t])$

$= v_k[t-1] * (1-\alpha_{ik}[t-1]) * (1-\alpha_{jk}[t-1])$

implies $(1-\alpha_{jk}[t]) = (1-\alpha_{ik}[t-1]) * (1-\alpha_{jk}[t-1])$,

the semijoin reducibility of R_i |X| R_j with R_k will be

$\alpha_{jk}[t] = \alpha_{ik}[t-1] + \alpha_{jk}[t-1] - \alpha_{ik}[t-1] * \alpha_{jk}[t-1]$.

Lemma: R_k X| $(R_i$ |X| $R_j) \subseteq (R_k$ X| $R_i)$ |X| $(R_k$ X| $R_j)$
with equality if $Y_{ik} = Y_{jk}$.

Proof:

R_k X| $(R_i$ |X| $R_j)$

$= \{t \mid t \in (R_i |X| R_j)$ & $\exists\ s_0 \in R_k$ s.t.

$s_0[Y_{ik} U Y_{jk}] = t[Y_{ik} U Y_{jk}]\}$

$= \{t \mid \exists\ s_1 \in R_i, s_2 \in R_j, s_0 \in R_k$ s.t.

$s_1 = t[R_i]$ & $s_2 = t[R_j]$ &

$s_1[Y_{ik}] = t[Y_{ik}] = s_0[Y_{ik}]$

$s_2[Y_{jk}] = t[Y_{jk}] = s_0[Y_{jk}]\}$.

Also

$$(R_K\ X|\ R_i)\ |X|\ \ (R_K\ X|\ R_j)$$
$$=\{t|\ \exists\ s_3 \in R_K\ X|R_i, s_4 \in R_K\ X|R_j\ \ s.t.$$
$$s_3 = t[R_i]\ \&\ s_4 = t[R_j]\ \}$$
$$=\{t|\ \exists\ s_3 \in R_i, s_4 \in R_j, s_5, s_6 \in R_K\ \ s.t.$$
$$s_3 = t[R_i]\ \&\ s_4 = t[R_j]\ \&$$
$$s_3[Y_{iK}] = t[Y_{iK}] = s_5[Y_{iK}]$$
$$s_4[Y_{jK}] = t[Y_{jK}] = s_6[Y_{jK}]\}$$

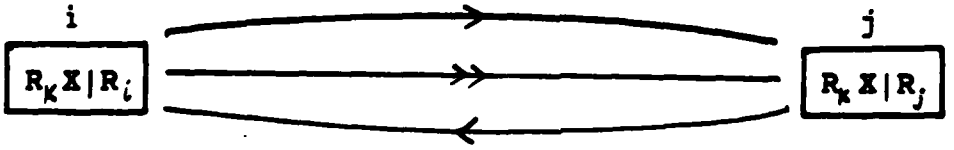
So we have

$$R_K X|\ (R_i\ |X|\ R_j) \subseteq (R_K\ X|\ R_i)\ |X|\ \ (R_K\ X|\ R_j).$$

Note that equality holds when $Y_{iK} = Y_{jK}$. Similarily, we approximate the size of $R_K X|\ (R_i\ |X|\ R_j)$ by the size of $(R_K\ X|\ R_i)\ |X|\ \ (R_K\ X|\ R_j)$.

Since the reducibilities between $R_K X|R_i$ and $R_K X|R_j$ are as follow:

$$1-\alpha_{ij}[t] = (1-\alpha_{Ki}[t-1]) * (1-\alpha_{ij}[t-1])$$

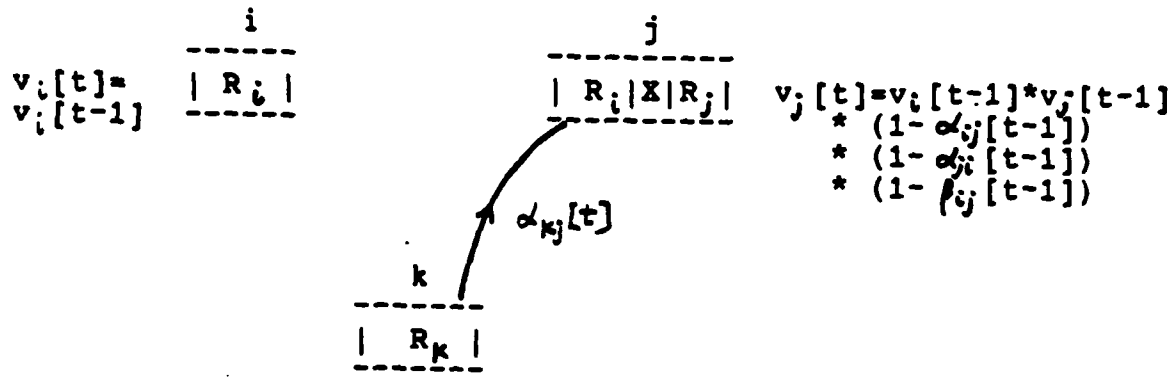


$$1-\alpha_{ji}[t] = (1-\alpha_{ji}[t-1]) * (1-\alpha_{Kj}[t-1])$$
$$1-\beta_{ij}[t] = (1-\beta_{ij}[t-1]) / (1-\alpha_{Kj}[t-1]) * (1-\alpha_{Ki}[t-1])$$

The size of $(R_K X|R_i)\ |X|\ (R_K X|R_j)$ will be

$$v_i[t] * v_j[t] * (1-\alpha_{ij}[t]) * (1-\alpha_{ji}[t]) * (1-\beta_{ij}[t])$$
$$= v_i[t-1] * (1-\alpha_{Ki}[t-1]) * v_j[t-1] * (1-\alpha_{Kj}[t-1])$$
$$* (1-\alpha_{ij}[t-1]) * (1-\alpha_{ji}[t-1]) * (1-\beta_{ij}[t-1]).$$

If at stage t, we have



By definition, the size of $R_k X|(R_i|X|R_j)$ is $v_j[t]*(1-\alpha_{kj}[t])$. So $v_j[t] * (1-\alpha_{kj}[t])$

$= v_i[t-1] * (1-\alpha_{ki}[t-1]) * v_j[t-1] * (1-\alpha_{kj}[t-1])$
$* (1-\alpha_{ij}[t-1]) * (1-\alpha_{ji}[t-1]) * (1-\beta_{ij}[t-1])$.

Since $v_j[t]$

$= v_i[t-1]* v_j[t-1]* (1-\alpha_{ij}[t-1]) * (1-\alpha_{ji}[t-1])$
$* (1-\beta_{ij}[t-1])$

implies $(1-\alpha_{kj}[t]) = (1-\alpha_{ki}[t-1]) * (1-\alpha_{kj}[t-1])$, so the semijoin reducibilities of $R_i |X| R_j$ with R_k will be

$$\alpha_{kj}[t]= \alpha_{kj}[t-1] + \alpha_{ki}[t-1] - \alpha_{kj}[t-1] * \alpha_{ki}[t-1].$$

We summarize the changing rules of semijoin reducibility of $R_h[t]$ to $R_k[t]$ after join operation b_{ij} as follows: (also see figure 2.2)

$$\alpha_{hk}[t]=\begin{cases} 0 & h=i;k=j \\ \alpha_{hk}[t-1] & h=j;k=i \\ \alpha_{hk}[t-1]+\alpha_{ik}[t-1]-\alpha_{hk}[t-1]*\alpha_{ik}[t-1] & h=j;k\neq i,j \\ \alpha_{hk}[t-1]+\alpha_{hi}[t-1]-\alpha_{hk}[t-1]*\alpha_{hi}[t-1] & k=j;h\neq i,j \\ \alpha_{hk}[t-1] & \text{otherwise} \end{cases}$$

Next, we consider the changing rules of join reducibilities after operation $p_t = b_{ij}$.(see figure 2.2) After performing $R_i |X|> R_j$ (i.e. b_{ij}) at stage t-1, the state at node i is $R_i[t] = R_i[t-1]$. The state at node j is R_i [t-1] $|X| R_j[t-1]$. $\alpha_{ij}[t]$ changes to 0 and $\alpha_{ji}[t]$ is the same as $\alpha_{ji}[t-1]$. Assume $\beta_{ij}[t]$ changes from $\beta_{ij}[t-1]$. We illustrate in the following figure.

At stage t,

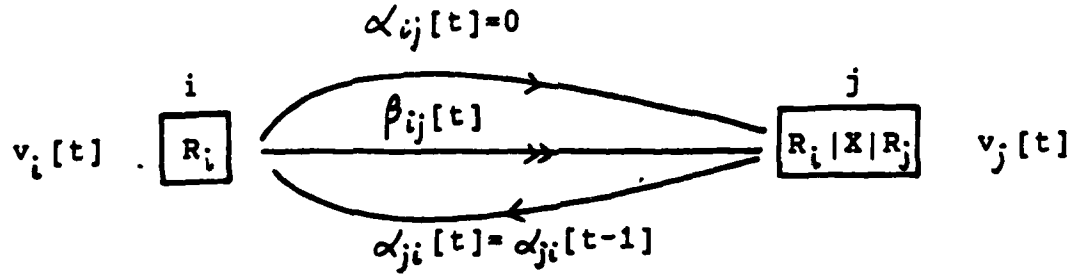


Figure 2.3 reducibilities after join operation $R_i |X|> R_j$

If we perform join b_{ij} again at a future stage, the resulting state at node j is the same because $R_i |X| (R_i |X| R_j)$ $= R_i |X| R_j$. Correspondingly, the estimated sizes of the derived relations must be the same.

That is $v_i[t]^* v_j[t]^* (1-\alpha_{ij}[t])^* (1-\alpha_{ji}[t])^* (1-\beta_{ij}[t]) = v_j$ [t].

Because $\alpha_{ij}[t]=0$ and $\alpha_{ji}[t] = \alpha_{ji}[t-1]$, we have

$$v_i[t]^* (1-\alpha_{ji}[t-1])^* (1-\beta_{ij}[t]) = 1$$

and resulting $\beta_{ij}[t] = 1 - 1/(v_i[t]^* (1-\alpha_{ji}[t-1]))$

$$= 1 - 1/(v_i[t-1]^* (1-\alpha_{ji}[t-1])),$$

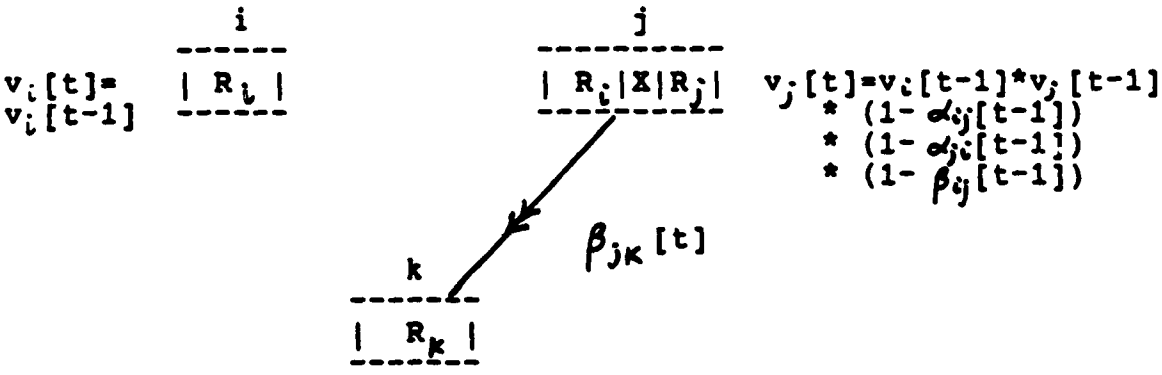
if $v_i[t-1]^* (1- \alpha_{ji}[t-1]) \neq 0$.

If $v_i[t-1]^* (1- \alpha_{ji}[t-1])=0$, then we set $\beta_{ij}[t]=1$.

Thus we have

$$\beta_{ij}[t]=\begin{cases} 1- 1/(v_i[t-1]^*(1- \alpha_{ji}[t-1])) & \text{if } v_i[t-1]^*(1- \alpha_{ji}[t-1])\neq 0. \\ 1 & \text{if } v_i[t-1]^*(1- \alpha_{ji}[t-1])=0. \end{cases}$$

Next, we consider the changing rule of join reducibility $\beta_{jk}[t]$. We assume at stage t,



We first look at another way of interpreting join reducibility. Let Y be the set of common attributes of R_i and R_j, W be the set of attributes in R_i -Y and Z be the set of attributes in R_j -Y, i.e. $R_i=\{Y,W\}$ and $R_j=\{Y,Z\}$. After two semijoin operations a_{ij} and a_{ji}, we assume

M = the number of common elements in Y columns.

Associated with each common element y_p in Y columns, let

N_{ip}=the number of w in columns W for y_p in R_i,

N_{jp}=the number of z in columns Z for y_p in R_j.

Thus, the size of R_i, $v_i = \sum_{p=1}^{M} N_{ip}$ and the size of R_j, v_j

$= \sum_{p=1}^{M} N_{2p}$. If we join R_i with R_j , $R_i|X|R_j$, the actual size of the relation $R_i|X|R_j$ will be $\sum_{p=1}^{M} N_{ip} * N_{jp}$. By the definition of join reducibility, the size of $R_i|X|R_j$ is

$$(\sum_{p=1}^{M} N_{ip})*(\sum_{p=1}^{M} N_{jp})*(1- \beta_{ij}).$$

Thus, we have

$$1- \beta_{ij} = (\sum_{p=1}^{M} N_{ip} * N_{jp})/((\sum_{p=1}^{M} N_{ip})*(\sum_{p=1}^{M} N_{jp})).$$

If $N_{ip} = N_{iq}$ and $N_{jp} = N_{jq}$ for all p, q, then

$$1- \beta_{ij} = 1/M$$

= 1/number of common elements in Y columns.

Suppose R_K is another relation with attributes {Y,U}, i.e. $Y_{ij} = Y_{jK} = Y_{iK} = Y$, and R_K has the same set of common elements in columns Y as of R_i and R_j. After performing all possible semijoin operations, R_i, R_j and R_K will have the same M common elements remain in columns Y. Let

N_{Kp}=the number of u in columns U for y_p in R_K .

The size of R_K will be $\sum_{p=1}^{M} N_{Kp}$. If we join $R_i|X|R_j|X|R_K$, the actual size of $R_i|X|R_j|X|R_K$ will be

$$\sum_{p=1}^{M} N_{ip} * N_{jp} * N_{Kp} \quad . \qquad (1)$$

If we join $R_i|X|R_j$ at node j, the size of $R_i|X|R_j$ is $\sum_{p=1}^{M} N_{ip} * N_{jp}$. Let the join reducibility of $R_i|X|R_j$ with R_K be β'_{jK} . If we join $R_i|X|R_j$ with R_K at node k, by definition we have the size of $R_i|X|R_j|X|R_K$ be

$$(\sum_{p=1}^{M} N_{ip} * N_{jp})*(\sum_{p=1}^{M} N_{Kp})*(1- \beta'_{jK}). \qquad (2)$$

Formulas (1) and (2) should be equal. Thus, we have

$$1- \beta'_{jK} = (\sum_{p=1}^{M} N_{ip} * N_{jp} * N_{Kp})/((\sum_{p=1}^{M} N_{ip} * N_{jp})*(\sum_{p=1}^{M} N_{Kp})).$$

$$= (\sum_{p=1}^{M} N_{ip} * N_{jp} * N_{Kp})*(\sum_{K=1}^{M} N_{Kp})/$$

$$((\sum_{p=1}^{M} N_{ip} * N_{jp}) * (\sum_{k=1}^{M} N_{kp}\ ^{*})).$$

If $N_{ip} = N_{iq}$, $N_{jp} = N_{jq}$, and $N_{kp} = N_{kq}$ for all p and q, then

$1-\beta'_{jk}$

$$= (\sum_{p=1}^{M} N_{ip} * N_{kp}) * (\sum_{p=1}^{M} N_{jp}) * (\sum_{p=1}^{M} N_{kp}) /$$
$$((\sum_{p=1}^{M} N_{ip}) * (\sum_{p=1}^{M} N_{kp}) * (\sum_{p=1}^{M} N_{ip}) * (\sum_{p=1}^{M} N_{kp})).$$

$$= (\sum_{p=1}^{M} N_{ip} * N_{kp}) * M * (\sum_{p=1}^{M} N_{jp} * N_{kp}) /$$
$$((\sum_{p=1}^{M} N_{ip}) * (\sum_{p=1}^{M} N_{kp}) * (\sum_{p=1}^{M} N_{jp}) * (\sum_{p=1}^{M} N_{kp})).$$

$$= M * (1-\beta_{ik}) * (1-\beta_{jk}).$$

In general, we let $M = \min_{p,q} \{ 1/(1-\beta_{qp}) \}$, i.e. the smallest number of common elements in joining columns of all pair of relations R_p and R_q. Thus, we approximate the join reducibility of $R_i |X| R_j$ with R_k by

$$1- M * (1-\beta_{ik}) * (1-\beta_{jk}).$$

Now we back to the changing rule of join reducibility $\beta_{jk}[t]$ after semijoin operation a_{ij}. Following the assumption of $Y_{ij} = Y_{jk}$, if we want to still have $\beta_{jk}[t]$ defined by pairwise formula

$$v_j[t] * v_k[t] * (1-\alpha_{jk}[t]) * (1-\alpha_{kj}[t]) * (1-\beta_{jk}[t]),$$

then $1-\beta_{jk}[t]$ will still be the reciprocal of the distinct elements in the joining columns, and after a_{ij}

$1-\beta_{jk}[t] = (1-\beta_{jk}[t-1])/(1-\alpha_{ij}[t-1])$. Thus, we approximate the join reducibility of $R_i X|R_j$ with R_k by

$$\beta_{jk}[t] = \max\{ 1-(1-\beta_{jk}[t-1])/(1-\alpha_{ij}[t-1]), 0\}.$$

In the case of n relations $\{R_i\}_{i=1}^{n}$, for which the semijoin and join reducibilities are $\{\alpha_{ij}, \beta_{ij}\}$ and the number of rows of relation R_i is v_i, by the above

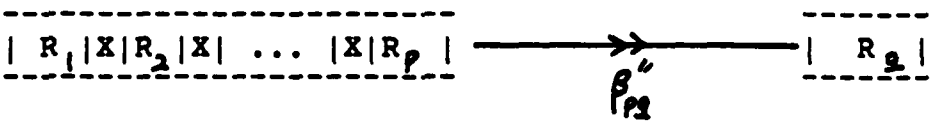
reducibility changing rules, we have the following results.

Lemma: For relations $R_1 |X| R_2 |X| \ldots . |X| R_p$ at node p and R at node q, let α''_{pq} , α''_{qp} and β''_{pq} be their corresponding semijoin and join reducibilities. They satisfy the following formulas.

(1) $1- \alpha''_{pq} = \prod_{h=1}^{p} (1- \alpha_{hq})$

(2) $1- \alpha''_{qp} = \prod_{h=1}^{p} (1- \alpha_{qh})$

(3) $1- \beta''_{pq} = M^{(p-1)} * \prod_{h=1}^{p} (1- \beta_{hq})$.



Proof: We prove this lemma by induction on p.

For p=2, by the assumption of semijoin and join reducibility changing rules, we have α''_{2q} , α''_{q2} and β''_{2q} of $R_1 |X| R_2$ with R_q satisfy

$1- \alpha''_{2q} = \prod_{h=1}^{2} (1- \alpha_{hq})$

$1- \alpha''_{q2} = \prod_{h=1}^{2} (1- \alpha_{qh})$

$1- \beta''_{2q} = M * \prod_{h=1}^{2} (1- \beta_{hq})$

for $q \neq 1,2$.

By the induction assumption, we have the semijoin and join reducibilities $\alpha''_{p-1,q}$, $\alpha''_{q,p-1}$ and $\beta''_{p-1,q}$ of $R_1 |X| R_2 |X| \ldots |X| R_{p-1}$ with R_q satisfy

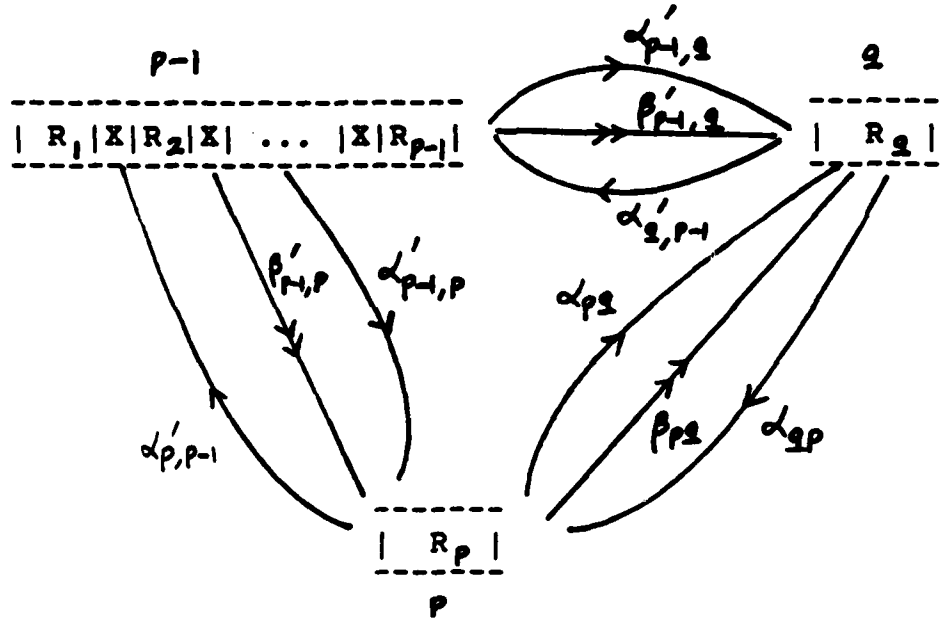
$1- \alpha''_{p-1,q} = \prod_{h=1}^{p-1} (1- \alpha_{hq})$

$1- \alpha''_{q,p-1} = \prod_{h=1}^{p-1} (1- \alpha_{qh})$

$1- \beta''_{p-1,q} = M^{p-2} * \prod_{h=1}^{p-1} (1- \beta_{hq})$

for q ≠ 1,2,...,p-1.

For the case of p,



Let the semijoin and join reducibilities of $R_1 |X| R_2 |X| \ldots |X| R_{p-1}$ with R_p be $\alpha'_{p-1,p}$, $\alpha'_{p,p-1}$ and $\beta'_{p-1,p}$ and the semijoin and join reducibilities of $R_1 |X| R_2 |X| \ldots |X| R_{p-1}$ with R_q be $\alpha'_{p-1,q}$, $\alpha'_{q,p-1}$ and $\beta'_{p-1,q}$. By the induction assumption, they satisfy

$$1- \alpha'_{p-1,q} = \prod_{h=1}^{p-1} (1- \alpha_{hq})$$

$$1- \alpha'_{q,p-1} = \prod_{h=1}^{p-1} (1- \alpha_{qh})$$

$$1- \beta'_{p-1,q} = M^{p-2} * \prod_{h=1}^{p-1} (1- \beta_{hq})$$

and

$$1- \alpha'_{p-1,p} = \prod_{h=1}^{p-1} (1- \alpha_{hp})$$

$$1- \alpha'_{p,p-1} = \prod_{h=1}^{p-1} (1- \alpha_{ph})$$

$$1- \beta'_{p,p-1} = M^{p-2} * \prod_{h=1}^{p-1} (1- \beta_{hp})$$

for q ≠ 1,2,...,p-1.

Also, we know the semijoin and join reducibilities

of R_p and R_q be α_{pq}, α_{qp} and β_{pq}. If we perform join operation $b_{p-1,p}$, i.e. $(R_1 |X| R_2 |X| \ldots |X| R_{p-1}) |X| > R_p$, then by the assumption of reducibility changing rules, we have the semijoin and join reducibilities α''_{pq}, α''_{qp} and β''_{pq} of $R_1 |X| R_2 |X| \ldots |X| R_p$ at node p and R_q at node q satisfy

$$1- \alpha_{pq}'' = (1- \alpha'_{p-1,q})*(1- \alpha_{pq}) = \prod_{h=1}^{p} (1- \alpha_{hq})$$

$$1- \alpha_{qp}'' = (1- \alpha'_{q,p-1})*(1- \alpha_{qp}) = \prod_{h=1}^{p} (1- \alpha_{qh})$$

$$1- \beta_{pq}'' = M * (1- \beta'_{p-1,q})*(1- \beta_{pq}) = M * M^{p-2} * \prod_{h=1}^{p-1} (1- \beta_{hq})*(1- \beta_{pq}) = M^{p-1} * \prod_{h=1}^{p} (1- \beta_{hq})$$

for $q \neq 1,2,\ldots,p$.

The lemma follows.

Lemma: The estimated size of relation $R_1 |X| R_2 |X| \ldots |X| R_n$ is

$$M^{\frac{(n-1)(n-2)}{2}} * \prod_{h=1}^{n} v_h * \prod_{\substack{h,k=1 \\ h \neq k}}^{n} (1- \alpha_{hk}) * \prod_{\substack{h,k=1 \\ h<k}}^{n} (1- \beta_{hk}).$$

Proof: We prove this lemma by induction on n.

For n=2, by the definition, the size of relation $R_1 |X| R_2$ is

$$v_1 * v_2 * (1- \alpha_{12})*(1- \alpha_{21})*(1- \beta_{12}).$$

By the induction assumption, we have the size of $R_1 |X| R_2 |X| \ldots |X| R_{n-1}$ be

$$M^{\frac{(n-2)(n-3)}{2}} * \prod_{h=1}^{n-1} v_h * \prod_{\substack{h,k=1 \\ h\neq k}}^{n-1} (1-\alpha_{hk}) * \prod_{\substack{h,k=1 \\ h<k}}^{n-1} (1-\beta_{hk}).$$

For the case of n, by the above lemma and induction assumption, we have the size of $R_1 |X| R_2 |X| \ldots |X| R_{n-1}$ be

$$M^{\frac{(n-2)(n-3)}{2}} * \prod_{h=1}^{n-1} v_h * \prod_{\substack{h,k=1 \\ h\neq k}}^{n-1} (1-\alpha_{hk}) * \prod_{\substack{h,k=1 \\ h<k}}^{n-1} (1-\beta_{hk}),$$

and the size of R_n be v_n. The semijoin and join reducibilities $\alpha'_{n-1,n}$, $\alpha'_{n,n-1}$ and $\beta'_{n-1,n}$ of $R_1 |X| R_2 |X| \ldots |X| R_{n-1}$ at node n-1 and R_n at node n satisfy

$$1-\alpha'_{n-1,n} = \prod_{h=1}^{n-1} (1-\alpha_{hn})$$

$$1-\alpha'_{n,n-1} = \prod_{h=1}^{n-1} (1-\alpha_{nh})$$

$$1-\beta'_{n,n-1} = M^{n-2} * \prod_{h=1}^{n-1} (1-\beta_{hk})$$

Thus, if we join $R_1 |X| R_2 |X| \ldots |X| R_{n-1}$ with R_n, the size of the resulting relation $R_1 |X| R_2 |X| \ldots |X| R_n$ will be

$$M^{\frac{(n-2)(n-3)}{2}} * \prod_{h=1}^{n-1} v_h * \prod_{\substack{h,k=1 \\ h\neq k}}^{n-1} (1-\alpha_{hk}) * \prod_{\substack{h,k=1 \\ h<k}}^{n-1} (1-\beta_{hk})$$

$$* v_n * (1-\alpha'_{n-1,n}) * (1-\alpha'_{n,n-1}) * (1-\beta'_{n,n-1})$$

$$= M^{\frac{(n-1)(n-2)}{2}} * \prod_{h=1}^{n} v_h * \prod_{\substack{h,k=1 \\ h\neq k}}^{n} (1-\alpha_{hk}) * \prod_{\substack{h,k=1 \\ h<k}}^{n} (1-\beta_{hk}).$$

The lemma follows.

we summarize the changing rules of join reducibilities of $R_h[t]$ to $R_k[t]$ after operation $p_t = b_{ij}$ as follows:

$$\beta_{jk}[t] = 1-M*(1-\beta_{ik}[t-1])*(1-\beta_{jk}[t-1]) \quad \forall\ k\neq i,j$$

$$\beta_{ij}[t] = \begin{cases} 1-1/(v_i[t-1]*(1-\alpha_{ji}[t-1])) & \text{if } v_i[t-1]*(1-\alpha_{ji}[t-1])\neq 0. \\ 1 & \text{if } v_i[t-1]*(1-\alpha_{ji}[t-1])=0. \end{cases}$$

$\beta_{hk}[t]=\beta_{hk}[t-1]$ otherwise

$\beta_{hk}[t]=\beta_{kh}[t]$ $\forall$ h,k

Theorem:

$0\leq \alpha_{ij}[t]\leq 1$ and $0\leq \beta_{ij}[t]\leq 1$, for all i and j and any time t.

Proof:

If $0\leq a\leq 1$ and $0\leq b\leq 1$ then $0\leq (a-1)(b-1)\leq 1$ implies $0\leq a+b-a\cdot b\leq 1$. We prove this theorem by induction on t. At t=0, the initial values $\alpha_{ij}[0]$ and $\beta_{ij}[0]$ are set to between 0 and 1. By the induction assumption at t-1, we have $0\leq \alpha_{ij}[t-1]\leq 1$ and $0\leq \beta_{ij}[t-1]\leq 1$.

At stage t, if $\alpha_{ij}[t]$ is equal to 0 or $\alpha_{ij}[t-1]$, or $\beta_{ij}[t]$ is equal to 1 or $\beta_{ij}[t-1]$, then they still between 0 and 1.

If $\alpha_{ij}[t]=\alpha_{ij}[t-1]+\alpha_{ik}[t-1]-\alpha_{ij}[t-1]*\alpha_{ik}[t-1]$ then by the induction assumption we have $0\leq \alpha_{ij}[t-1]\leq 1$ and $0\leq \alpha_{ik}[t-1]\leq 1$,

which implies $0\leq \alpha_{ij}[t]\leq 1$.

For $\beta_{jk}[t]=1-M*(1-\beta_{ik}[t-1])*(1-\beta_{jk}[t-1])$, because $M=\min_{p,q}\{1-(1-\beta_{pq}[0])\}$ implies $1/M=\max\{(1-\beta_{pq}[0]\}$, thus we have $M*(1-\beta_{pq}[t-1])\leq 1$. This implies $0\leq M*(1-\beta_{ik}[t-1])*(1-\beta_{jk}[t-1])\leq 1$, i.e. $0\leq \beta_{jk}[t]\leq 1$.

If $v_i[t-1]*(1-\alpha_{ji}[t-1])\neq 0$, then $v_i[t-1]*(1-\alpha_{ji}[t-1])$ is greater than or equal to 1.

Thus, $\beta_{ij}[t]=1-1/(v_i[t-1]*(1-\alpha_{ji}[t-1]))$ is between

0 an 1. These prove $0 \leq \alpha_{ij}[t] \leq 1$ and $0 \leq \beta_{ij}[t] \leq 1$. The theorem follows.

We note to readers that for the case of three relations R_1, R_2 and R_3 having the same joining columns, i.e. $Y=Y_{12}=Y_{13}=Y_{23}$, $a_{31}a_{12}$ and $a_{31}a_{12}a_{32}$ will actually produce the same results at node 2. But, under this model, the estimated sizes of that result by following these two strategies are different. In general, the strategy $a_{31}a_{12}a_{32}$ is more time comsuming than the strategy $a_{31}a_{12}$ and usually we do not use it.

Since the semijoin of R_i X|> R_j requires the projection of R_i over Y_{ij} and sending the result to to node j, the projection of R_i over $R_i \cap R_j$ may reduce the number of rows of R_i by eliminating multiple copies of tuples that are the same over $R_i \cap R_j$. We want to estimate the number of tuples of a relation after projection.

Let W be the set of attributes in $R_i - Y_{ij}$ and Z be the set of attributes in $R_j - Y_{ij}$, i.e. $R_i = \{ W, Y_{ij} \}$ and $R_j = \{ Y_{ij}, Z \}$.

Let N(YW)= number of w values in R_i [W] per y and

N(YZ) = number of z values in R_j [Z] per y.

N(1)=number of y in $R_i[Y_{ij}]$.

N(2)=number of y in $R_j[Y_{ij}]$.

N(0)=number of common y in $R_i[Y_{ij}]$ and $R_j[Y_{ij}]$.

Thus v_i = N(1) * N(YW) , v_j = N(2) * N(YZ).

After semijoin R_i X|R_j and R_j X|R_i , then

v_i = N(0) * N(YW) = v_i * (1-α_{ji}) and

v_j = N(0) * N(YZ) = v_j * (1-α_{ij}).

The size of R_i |X|R_j is N(Join)= N(0) * N(YW) * N(YZ).

According to the definition of reducibilities, we have

N(Join) = v_i * v_j* (1-α_{ij})* (1-α_{ji})* (1-β_{ij})

= N(1) * N(YW) * N(2) * N(YZ)

* (1-α_{ij})* (1-α_{ji})* (1-β_{ij})

= N(0) * N(YW) * N(0) * N(YZ) * (1-β_{ij})

= N(0) * N(YW) * N(YZ)

This implies N(0) = 1 /(1-β_{ij}).

Since N(0) * N(YW) = N(1) * N(YW) * (1-α_{ji}),

N(1)= 1 /((1-α_{ij}) * (1-β_{ij})).

Similarily, N(2)= 1 /((1-α_{ij}) * (1-β_{ij})).

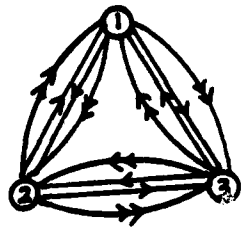
Thus, we will estimate the number of rows after projection of R_i over Y_{ij} by

1 /((1-α_{ij}) * (1-β_{ij})).

If α_{ij} =1, then R_i and R_j do not have common values in the joining column of R and R . then α_{ji} = β_{ij} =1. In fact, either one of α_{ij} , α_{ji} or β_{ij} equal to 1 will implies another two also equal to one. If one of them is equal to 1 then the result is empty.

Example:

Suppose we have three relations R_1 ,R_2 and R_3 , where $R_i \cap R_j = \phi$ and $R_i \nsubseteq R_j \forall$ i,j; and suppose $v_i[0]$, $\alpha_{ij}[0]$, $\beta_{ij}[0]$ are given. Then the processing graph will be:



Let $P_1 = a_{32}\, b_{23}\, b_{31}$ and $P_2 = a_{12}\, a_{13}\, b_{31}\, b_{21}$ be two programs. Both two programs P_1 and P_2 will produce the same results $R_1[0]$ |X| $R_2[0]$ |X| $R_3[0]$ at site 1. (see Figure 2.4) By the rules of estimating the size of the derived relation, the estimate sizes of $R_1[0]$ |X| $R_2[0]$ |X| $R_3[0]$ derived by these two programs will be the same. Which is $M * \prod_{i=1}^{3} v_i[0] * \prod_{\substack{i,j=1 \\ i<j}}^{3} (1-\alpha_{ij}[0]) * \prod_{\substack{i,j=1 \\ i<j}}^{3} (1-\beta_{ij}[0])$, where $M = \min_{p,q}\{1-(1-\beta_{pq}[0])\}$.

For $P_1 = a_{32}\, b_{23}\, b_{31}$,

After performing a_{32}

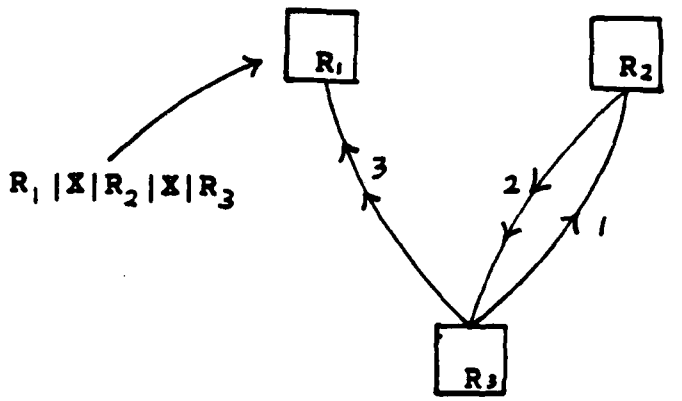
$v_1[1] = v_1[0]$

$v_2[1] = v_2[0] * (1-\alpha_{32}[0])$

$v_3[1] = v_3[0]$

P_1 : a_{32} b_{23} b_{31}

$R_3|X|>R_2$ $R_2|X|>R_3$ $R_3|X|>R_1$



P_2 : a_{12} a_{13} b_{31} b_{21}

$R_1|X|>R_2$ $R_1|X|>R_3$ $R_3|X|>R_1$ $R_2|X|>R_1$

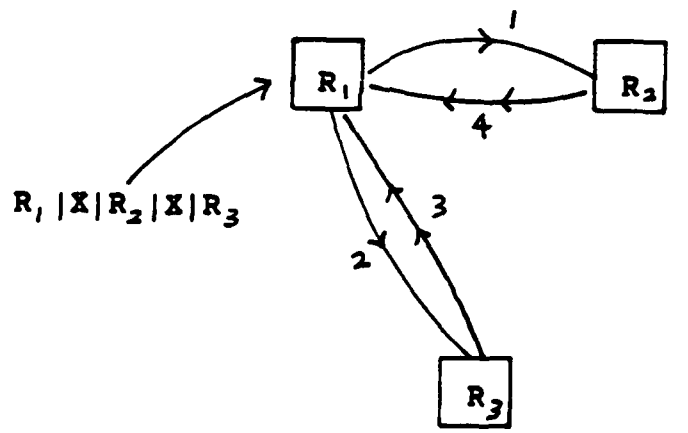


Figure 2.4 two programs P_1 and P_2.

$\alpha_{32}[1]=0$

$\alpha_{23}[1]=\alpha_{23}[0]$

$\alpha_{13}[1]=\alpha_{13}[0]$

$\alpha_{31}[1]=\alpha_{31}[0]$

$\alpha_{12}[1]=\alpha_{12}[0]$

$\alpha_{21}[1]=\alpha_{21}[0]+\alpha_{32}[0]-\alpha_{21}[0]*\alpha_{32}[0]$

$\beta_{12}[1]=1-(1-\beta_{12}[0])/(1-\alpha_{32}[0])$

$\beta_{ij}[1]=\beta_{ij}[0]$ otherwise

After performing b_{23}

$v_1[2]=v_1[1]$

$v_2[2]=v_2[1]$

$v_3[2]=v_3[1]*v_2[1]*(1-\alpha_{32}[1])*(1-\alpha_{23}[1])*(1-\beta_{23}[1])$

$=v_3[0]*v_2[0]*(1-\alpha_{32}[0])*(1-\alpha_{23}[0])*(1-\beta_{23}[0])$

$\alpha_{32}[2]=0$

$\alpha_{23}[2]=0$

$\alpha_{12}[2]=\alpha_{12}[1]$

$\alpha_{21}[2]=\alpha_{21}[1]$

$\alpha_{13}[2]=\alpha_{13}[0]+\alpha_{12}[0]-\alpha_{13}[0]*\alpha_{12}[0]$

$\alpha_{31}[2]=\alpha_{31}[0]+\alpha_{21}[0]-\alpha_{31}[0]*\alpha_{21}[0]$

$\beta_{23}[2]=1-1/v_2[1]$

$\beta_{12}[2]=\beta_{12}[1]=\beta_{12}[0]$

$\beta_{13}[2]=1-M*(1-\beta_{13}[0])*(1-\beta_{12}[0])$

After performing b_{31}

$v_2[3]=v_2[2]$

$v_3[3]=v_3[2]$

$v_1[3]=v_1[2]*v_3[2]*(1-\alpha_{13}[2])*(1-\alpha_{31}[2])*(1-\beta_{31}[2])$

$$= M^* \prod_{i=1}^{3} v_i[0]^* \prod_{\substack{i,j=1 \\ i\neq j}}^{3} (1-\alpha_{ij}[0])^* \prod_{\substack{i,j=1 \\ i<j}}^{3} (1-\beta_{ij}[0])$$

For $P_2 = a_{12}\, a_{13}\, b_{31}\, b_{21}$,

After performing a_{12}

$v_1[1]=v_1[0]$

$v_2[1]=v_2[0]*(1-\alpha_{12}[0])$

$v_3[1]=v_3[0]$

$\alpha_{12}[1]=0$

$\alpha_{21}[1]=\alpha_{21}[0]$

$\alpha_{13}[1]=\alpha_{13}[0]$

$\alpha_{31}[1]=\alpha_{31}[0]$

$\alpha_{23}[1]=\alpha_{23}[0]+\alpha_{12}[0]-\alpha_{23}[0]*\alpha_{12}[0]$

$\alpha_{32}[1]=\alpha_{32}[0]$

$\beta_{23}[1]=1-(1-\beta_{23}[0])/(1-\alpha_{12}[0])$

$\beta_{hk}[1]=\beta_{hk}[0]$ otherwise

After performing a_{13}

$v_1[2]=v_1[0]$

$v_2[2]=v_2[0]*(1-\alpha_{12}[0])$

$v_3[2]=v_3[0]*(1-\alpha_{13}[0])$

$\alpha_{12}[2]=0$

$\alpha_{13}[2]=0$

$\alpha_{21}[2]=\alpha_{21}[0]$

$\alpha_{31}[2]=\alpha_{31}[0]$

$\alpha_{23}[2]=\alpha_{23}[0]$

$\alpha_{32}[2] = \alpha_{32}[0] + \alpha_{13}[0] - \alpha_{32}[0] * \alpha_{13}[0]$

$\beta_{23}[2] = 1 - (1 - \beta_{23}[1])/(1 - \alpha_{13}[0])$

$= 1 - (1 - \beta_{23}[0])/(1 - \alpha_{13}[0]) * (1 - \alpha_{12}[0])$

$\beta_{hk}[2] = \beta_{hk}[0]$ otherwise

After performing b_{31}

$v_1[3] = v_1[0] * v_3[0] * (1 - \alpha_{13}[0]) * (1 - \alpha_{13}[0]) * (1 - \beta_{13}[0])$

$v_2[3] = v_2[2]$

$v_3[3] = v_3[2]$

$\alpha_{31}[3] = 0$

$\alpha_{13}[3] = \alpha_{13}[2] = 0$

$\alpha_{23}[3] = \alpha_{23}[2]$

$\alpha_{32}[3] = \alpha_{32}[2]$

$\alpha_{21}[3] = \alpha_{21}[0] + \alpha_{23}[0] - \alpha_{21}[0] * \alpha_{23}[0]$

$\alpha_{12}[3] = \alpha_{12}[2] + \alpha_{32}[2] - \alpha_{12}[2] * \alpha_{32}[2]$

$\beta_{31}[3] = 1 - 1/v_3[2]$

$\beta_{32}[3] = \beta_{32}[2] = \beta_{32}[0]$

$\beta_{12}[3] = 1 - M * (1 - \beta_{12}[2]) * (1 - \beta_{32}[2])$

$= 1 - M * (1 - \beta_{12}[0]) * (1 - \beta_{32}[0]) / ((1 - \alpha_{13}[0]) * (1 - \alpha_{12}[0]))$.

After performing b_{21}

$v_2[4] = v_2[3]$

$v_3[4] = v_3[3]$

$v_1[4] = v_1[3] * v_2[3] * (1 - \alpha_{12}[3]) * (1 - \alpha_{21}[3]) * (1 - \beta_{12}[3])$

$= M * \prod_{i=1}^{3} v_i[0] * \prod_{\substack{i,j=1 \\ i \neq j}}^{3} (1 - \alpha_{ij}[0]) * \prod_{\substack{i,j=1 \\ i<j}}^{3} (1 - \beta_{ij}[0])$

In practical implementation, we can initially set α_{ij} and β_{ij} for each pair of relations by some number which we can intuitively guess at the time that the databases are implemented. At each node, we set some mechanisms to record the statistical information of semijoin reducibilities and join reducibilities. After the systems are running, we can update the number of tuples in each relation and the reducibilities of each pair of relations.

In processing a given query, we perform initial local processing (including projection and selection operations) and attribute renaming process first. Assume the size of relations $\{v_i\}_{i=1}^{n}$, reducibilities $\{\alpha_{ij}, \beta_{ij}\}$ and the number of tuples of relations associated in solving the given query after local processing, $v_i[0]$, for relation R_i are known. The reducibilities $\{\alpha_{ij}[0], \beta_{ij}[0]\}$ of the resulting relations associated with this query will depend on the local processing. In the following, we will derive formulas for $\alpha_{ij}[0]$ and $\beta_{ij}[0]$.

Let W be the set of attributes in $R_i - Y_{ij}$ and Z be the set of attributes in $R_j - Y_{ij}$, i.e. $R_i = \{W, Y_{ij}\}$ and $R_j = \{Y_{ij}, Z\}$. Let

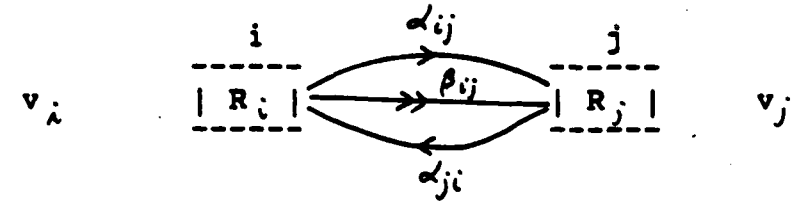
N(YW)= number of w values in $R_i[W]$ per y;

N(YZ) = number of z values in $R_j[Z]$ per y;

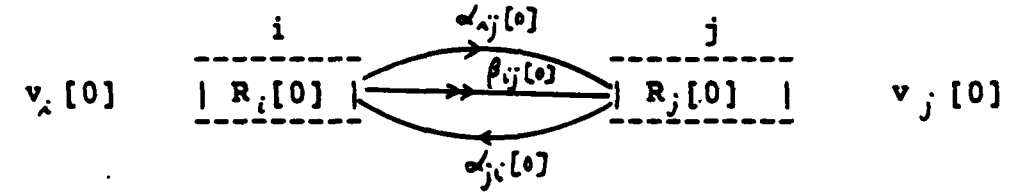
N(A)=number of common y in $R_i[Y_{ij}]$ and $R_j[Y_{ij}]$ before local processing;

N(B)=number of common y in $R_i[Y_{ij}]$ and $R_j[Y_{ij}]$ after local processing.

We illustrated as in the following figure:
Before local processng



After local processing



We assume the number of common elements of the joining domain being reduced is proportional to the size of relations being reduced,

i.e. $N(B) = N(A) * (v_i[0]/v_i) * (v_j[0]/v_j)$.

So $N(B) * N(YZ)$

$= N(A) * (v_i[0]/v_i) * (v_j[0]/v_j) * N(YZ)$

$= v_j * (1- \alpha_{ij}) * (v_i[0]/v_i) * (v_j[0]/v_j)$

$= v_j[0] * (1- \alpha_{ij}[0])$.

This implies $(1- \alpha_{ij}) * (v_i[0]/v_i) = 1- \alpha_{ij}[0]$.

So we have $\alpha_{ij}[0] = 1 - (1 - \alpha_{ij}) * (v_i[0]/v_i)$.

Similarily, we have $\alpha_{ji}[0] = 1 - (1 - \alpha_{ji}) * (v_j[0]/v_j)$.

Because $N(B) * N(YW) * N(YZ)$

$= N(A) * (v_i[0]/v_i) * (v_j[0]/v_j) * N(YW) * N(YZ)$

$= v_i * v_j * (1 - \alpha_{ij}) * (1 - \alpha_{ji}) * (1 - \beta_{ij})$
$* (v_i[0]/v_i) * (v_j[0]/v_j)$

$= v_i[0] * v_j[0] * (1 - \alpha_{ij}[0]) * (1 - \alpha_{ji}[0]) * (1 - \beta_{ij}[0])$

$= v_i[0] * v_j[0] * (1 - \alpha_{ij}) * (v_i[0]/v_i) * (1 - \alpha_{ji})$
$* (v_j[0]/v_j) * (1 - \beta_{ij}[0])$

This implies

$$(1 - \beta_{ij}) = (1 - \beta_{ij}[0]) * (v_i[0]/v_i) * (v_j[0]/v_j).$$

That is $\beta_{ij}[0] = \beta_{ij} + (1 - \beta_{ij}) * (1 - (v_i/v_i[0]) * (v_j/v_j[0]))$.

We choose $\beta_{ij}[0]$ to be $\max\{0, \beta_{ij} + (1 - \beta_{ij}) * (1 - (v_i/v_i[0]) * (v_j/v_j[0]))\}$. The set of numbers $\{v_i[0], \alpha_{ij}[0], \beta_{ij}[0]\}$ are the initial values for analyzing this given query.

2.4.4 Problem Formulation

In order to write down the mathematical formulation of the distributed query optimization problem, we need to know the cost function of each operation. From our previous assumption of a linear cost function, we can write down the cost function of operations at stage t.

The cost of operation a_{ij} will be

$$Cost(a_{ij}) = c_{ij} * (v_{Y_{ij}} * \sum_{A \in Y_{ij}} w(A))$$

and the cost of operation b_{ij} will be

$$Cost(b_{ij}) = c_{ij} * (v_i[t] * \sum_{A \in R_i} w(A)).$$

Based on the distributed query processing model we developed, the formulation of the distributed query optimization problem is as follow:

INPUT:

1. a distributed database schema $D=\{ R_1[0],\ldots,R_n[0]\}$
2. the width w(A) of each attribute A in U(D)
3. the number of rows , $v_i[0]$, of each relation R_i
4. the semijoin reducibility $\alpha_{ij}[0]$ of each pair of relations R_i & R_j with $\alpha_{ii}[0]=0$
5. the join reducibility $\beta_{ij}[0]$ of each pair of relations R_i & R_j with $\beta_{ii}[0]=0$ and $\beta_{ij}[0]= \beta_{ji}[0]$.

OBJECTIVE:

Find an optimal join-semijoin program to solve the natural join program.

Let $P=p_1 p_2 \ldots p_\ell$, then the problem can be written in the form:

$$\min_{P} \quad \sum_{t=1}^{\ell} \text{cost}(p_t)$$

s.t. $D[t]=f_1(D[t-1])$

$A[t]=f_2(A[t-1], B[t-1])$

$B[t]=f_3(A[t-1], B[t-1])$

$\underline{v}[t]=f_4(\underline{v}[t-1], A[t-1], B[t-1])$

and

$\underline{v}[0]$, $A[0]$, $B[0]$ are given.

Where $A[0]=[\alpha_{ij}[0]]$ and

$B[0]=[\beta_{ij}[0]]$ are

the initial reducibility matrices.

$\underline{v}[0]=[v_i[0]]$ is

the initial size of relation D .

$f_1(D[t-1])$ is the mapping from temporary database state at stage t-1 by the operation p_t to a new temporary database state at stage t. During the analysis of a query solution strategy, the database state is conceptually considered changing from one state to the other state by an operation. The real database state stored in the system does not change

unless update operations are really performed.

and f_2, f_3 and f_4 are mappings according the definition of reducibilities and its changing rules.

If $p_x = a_{ij}$, then

$$\underline{v}[t] = \begin{array}{c} 1 \\ \vdots \\ j \\ \vdots \\ k \\ \vdots \\ n \end{array} \begin{bmatrix} v_1[t-1] \\ \vdots \\ v_j[t-1] * (1-\alpha_{ij}[t-1]) \\ \vdots \\ v_k[t-1] \\ \vdots \\ v_n[t-1] \end{bmatrix}$$

A[t] =

$$\begin{array}{c} \\ 1 \\ \vdots \\ i \\ \vdots \\ k \\ \vdots \\ j \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & & i & & k & & j & & n \end{array} \\ \begin{bmatrix} 0 & \cdots & \alpha_{1i}[t-1] & \cdots & \alpha_{1k}[t-1] & \cdots & \alpha_{1j}[t-1] & \cdots & \alpha_{1n}[t-1] \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ \alpha_{i1}[t-1] & \cdots & 0 & \cdots & \alpha_{ik}[t-1] & \cdots & 0 & \cdots & \alpha_{in}[t-1] \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ \alpha_{k1}[t-1] & \cdots & \alpha_{ki}[t-1] & \cdots & 0 & \cdots & \alpha_{kj}[t-1] & \cdots & \alpha_{kn}[t-1] \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ \alpha_{j1}[t-1] & \cdots & \alpha_{ji}[t-1] & \cdots & \alpha_{jk}[t-1] + \alpha_{ij}[t-1] - \alpha_{jk}[t-1] * \alpha_{ij}[t-1] & \cdots & 0 & \cdots & \alpha_{jn}[t-1] \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ \alpha_{n1}[t-1] & \cdots & \alpha_{ni}[t-1] & \cdots & \alpha_{nk}[t-1] & \cdots & \alpha_{nj}[t-1] & \cdots & 0 \end{bmatrix} \end{array}$$

$B[t] =$

$$
\begin{array}{c}
 \\
1 \\
\vdots \\
i \\
\vdots \\
k \\
\vdots \\
j \\
\vdots \\
n
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc} 1 & & i & & k & & j & & n \end{array} \\
\begin{bmatrix}
0 & \cdots & \beta_{1i}[t-1] & \cdots & \beta_{1k}[t-1] & \cdots & \beta_{1j}[t-1] & \cdots & \beta_{1n}[t-1] \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
\beta_{i1}[t-1] & \cdots & 0 & \cdots & \beta_{ik}[t-1] & \cdots & \beta_{ij}[t-1] & \cdots & \beta_{in}[t-1] \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
\beta_{k1}[t-1] & \cdots & \beta_{ki}[t-1] & \cdots & 0 & \cdots & \beta_{kj}[t-1] & \cdots & \beta_{kn}[t-1] \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
\beta_{j1}[t-1] & \cdots & \beta_{ji}[t-1] & \cdots & 1-(1-\beta_{jk}[t-1])/(1-\alpha_{ij}[t-1]) & \cdots & 0 & \cdots & \beta_{jn}[t-1] \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
\beta_{n1}[t-1] & \cdots & \beta_{ni}[t-1] & \cdots & \beta_{nk}[t-1] & \cdots & \beta_{nj}[t-1] & \cdots & 0
\end{bmatrix}
\end{array}
$$

$$\text{cost}(p_t) = c_{ij} * \left(v_i[t-1] * \sum_{A \in Y_{ij}[t-1]} w(A) \right)$$

$$R_i[t] = R_i[t-1] \quad \forall\, i.$$

If $p_t = b_{ij}$, then

$$\underline{v}[t] = \begin{array}{c} \\ 1 \\ \vdots \\ j \\ \vdots \\ \vdots \\ k \\ \vdots \\ n \end{array} \begin{bmatrix} v_1[t-1] \\ \vdots \\ v_j[t-1] * (1-\alpha_{ij}[t-1]) * v_i[t-1] * (1-\alpha_{ji}[t-1]) * (1-\beta_{ij}[t-1]) \\ \vdots \\ v_K[t-1] \\ \vdots \\ v_n[t-1] \end{bmatrix}$$

$A[t] =$

$$\begin{array}{c} \\ 1 \\ \\ \\ \vdots \\ i \\ \vdots \\ k \\ \\ \\ \vdots \\ j \\ \\ \\ \\ \vdots \\ n \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & & i & & k & & j \end{array} \\ \begin{bmatrix} 0 & \cdots & \alpha_{1i}[t-1] & \cdots & \alpha_{1K}[t-1] & \cdots & \alpha_{1j}[t-1]+\alpha_{1i}[t-1]-\alpha_{ij}[t-1]*\alpha_{1i}[t-1] \\ \vdots & & \vdots & & \vdots & & \vdots \\ \alpha_{i1}[t-1] & \cdots & 0 & \cdots & \alpha_{iK}[t-1] & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ \alpha_{K1}[t-1] & \cdots & \alpha_{Ki}[t-1] & \cdots & 0 & \cdots & \alpha_{Kj}[t-1]+\alpha_{Ki}[t-1]-\alpha_{Kj}[t-1]*\alpha_{Ki}[t-1] \\ \vdots & & \vdots & & \vdots & & \vdots \\ \alpha_{j1}[t-1]+\alpha_{i1}[t-1]-\alpha_{j1}[t-1]*\alpha_{i1}[t-1] & \cdots & \alpha_{ji}[t-1] & \cdots & \alpha_{jK}[t-1]+\alpha_{iK}[t-1]-\alpha_{jK}[t-1]*\alpha_{iK}[t-1] & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ \alpha_{n1}[t-1] & \cdots & \alpha_{ni}[t-1] & \cdots & \alpha_{nK}[t-1] & \cdots & \alpha_{nj}[t-1]+\alpha_{ni}[t-1]-\alpha_{nj}[t-1]*\alpha_{ni}[t-1] \end{bmatrix} \end{array} \begin{array}{c} \\ n \\ \\ \\ \vdots \\ i \\ \vdots \\ k \\ \\ \\ \vdots \\ j \\ \\ \\ \\ \vdots \\ n \\ \\ \\ \end{array}$$

$$\begin{array}{c} \\ \end{array} \begin{array}{c} \\ \begin{bmatrix} \alpha_{1n}[t-1] \\ \vdots \\ \alpha_{in}[t-1] \\ \vdots \\ \alpha_{Kn}[t-1] \\ \vdots \\ \alpha_{jn}[t-1]+\alpha_{in}[t-1]-\alpha_{jn}[t-1]*\alpha_{in}[t-1] \\ \vdots \\ 0 \end{bmatrix} \end{array}$$

$B[t] =$

	1	i	k	j	n
1	0	$\beta_{1i}[t-1]$	$\beta_{1k}[t-1]$	$1-M^*(1-\beta_{1i}[t-1])*(1-\beta_{ij}[t-1])$	$\beta_{1n}[t-1]$
i	$\beta_{i1}[t-1]$	0	$\beta_{ik}[t-1]$	$1-1/(v_i[t-1]*(1-\alpha_{ji}[t-1]))$	$\beta_{in}[t-1]$
k	$\beta_{k1}[t-1]$	$\beta_{ki}[t-1]$	0	$1-M^*(1-\beta_{ki}[t-1])*(1-\beta_{kj}[t-1])$	$\beta_{kn}[t-1]$
j	$1-M^*(1-\beta_{ji}[t-1])*(1-\beta_{i1}[t-1])$	$1-1/(v_i[t-1]*(1-\alpha_{ji}[t-1]))$	$1-M^*(1-\beta_{jk}[t-1])*(1-\beta_{ik}[t-1])$	0	$1-M^*(1-\beta_{jn}[t-1])*(1-\beta_{in}[t-1])$
n	$\beta_{n1}[t-1]$	$\beta_{ni}[t-1]$	$\beta_{nk}[t-1]$	$1-M^*(1-\beta_{ni}[t-1])*(1-\beta_{nj}[t-1])$	0

$$M = \min_{p,q} \{1/(1-\beta_{pq}[0])\}$$

$$cost(p_x) = c_{ij} * (v_i[t-1] * \sum_{A \in Y_{ij}} w(A))$$

$$R_i[t] = R_i[t-1] \quad \forall\ i \neq j$$

$$R_j[t] = R_i[t-1] \cup R_j[t-1]$$

2.5 Extend The Model to Include Local Processing Cost

The underlying assumption of this chapter is that of communication cost dominance. This is ususlly true in the case of a large scale communication network. When databases are distributed in a local area network environment, the local processing costs also play a significant role in query processing because selection, projection and join operations sometime take a significant processing time to process the operations. In the case where local processing costs are comparable to communication costs, this model can be easily extended to cover the situation by providing a method for estimating the amount of processing time required and associating each node with a local processing cost. A method of estimating the local processing cost of selection, projection and join operations was studied by [SEL 79] and [KIM 80], etc. This cost depends on the method of implementing join operation and the available main-memory buffer space.

2.6 Conclusions

In this chapter, we considered the query processing problem in a distributed relational database envirnment, and we extended previous work to consider a larger class of solution strategies for equi-join query processing.

We have developed a mathematical model to compute the minimum communication cost of a join-semijoin program for processing a given equi-join query. We defined a query processing graph for each equi-join query and characterize the set of join-semijoin programs which solve this query. A rule for estimating the size of the derived relation is assumed. With the assumption of communication cost dominance, when the cost functions are linear in the size of data transmission, an optimization problem for distributed query processing is formulated and solved. This model can be extended to the case where local processing costs are significant and nonnegligible by associating each node with a local processing cost and providing a method for estimating local processing cost.

Although the model is based on processing the class of equi-join queries which is a subset of complete relatioanl calculus language, it is a rich and large class of queries in practice.

In a general query processing, we can divide the clauses in the qualification of a query into two sets: The set of equality clauses and the set of inequality clauses. Usually the set of inequality clauses is very small. We can either process the set of inequality clauses by using the inequality joins and inequality semijoins first and leave the remaining equi-join query solving by using the model or vice versa. The other approach is to change all inequality

join clauses into equality join clauses by referencing the domains of join attributes.

A future research topic is to extend this model to cover a large class of inequality join queries by providing a method to measure the reducibilities and to estimate the size of the derived relations.

Chapter 3

Computation Complexity of Distributed Query Processing Problem

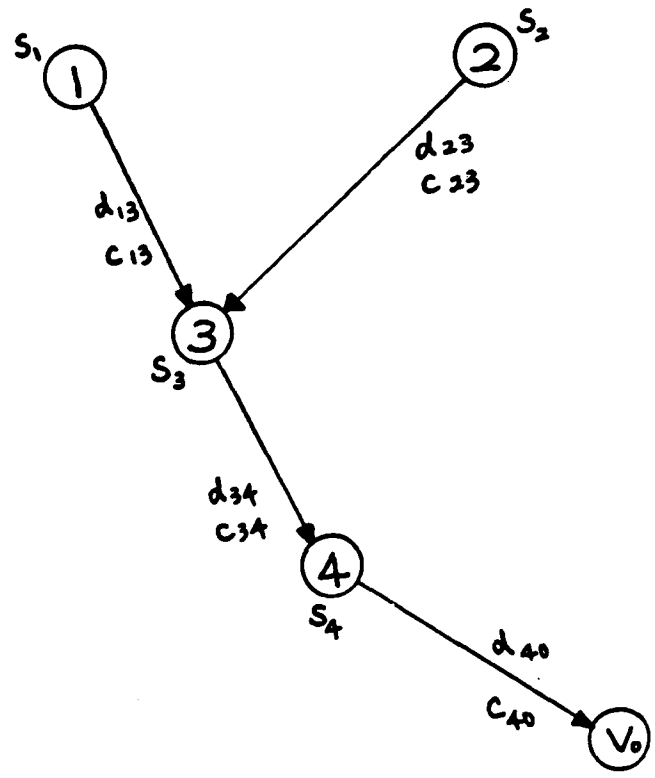
3.1 Introduction

In distributed database management systems, the efficiency of query processing has a strong influence on the performance of the systems. Query processing in a distributed system is different from that in a centralized system. In a distributed system, data are stored at computers which are geographically separated; hence query processing involves some local data processing and the necessary data transmission over communication links. Although both types of operations will introduce time delays, for a large network of databases the transmission delay plays the major role in the overall system performance. Many researchers and system developers have considered these facts and have derived ways of finding a distributed processing strategy for data processing and data transmissions. It is recognized that deriving an optimal distributed processing strategy, in the sense that some cost is minimized, is a very difficult problem and all algorithms that have been proposed to date are heuristic. No one has yet been able to show that finding an optimal query processing strategy in distributed databases is an intractable problem.

In the last chapter, under the assumption that data transmission costs dominate tha local processing costs, we developed a model for solving an equi-join query by a strategy employing a mixed sequence of joins and semijoins. The objective is to minimize the total data transmission costs for processing the query. If we perform all the possible semijoin operations, as in the strategy used in SDD-1 [GBWRR 81], the remaining problem becomes one of finding a routing strategy of sending required data to the site where the query was initiated with a minimum total transmission cost. In SDD-1, the system takes the simplest way by sending all the data to the query initiating node and processing at that node. In this chapter, we will consider the case where all semijoin reducibilities are equal to zero and the join reducibilities are not affected by join operations. We call this problem the query processing problem (QP). For our purpose, we formulate the problem in the following way.

Query processing problem (QP):

Given a complete directed graph G=(V,E); the size of data associated with each node i, s_i ; the unit communication cost of edge(i,j), c_{ij} ; and the join reducibility associated with edge(i,j), d_{ij} . Here d_{ij} has the same interpretation as $1-\beta_{ij}$ in chapter 2. All semijoin reducibilities α_{ij} in chapter 2 are equal to zero. The size of data at node j after the arrival of data from node i and



THE COST OF THIS SUBTREE IS

$$S_1 \cdot C_{13} + S_2 \cdot C_{23} + (S_1 \cdot S_2 \cdot S_3 \cdot d_{13} \cdot d_{23}) \cdot C_{34}$$

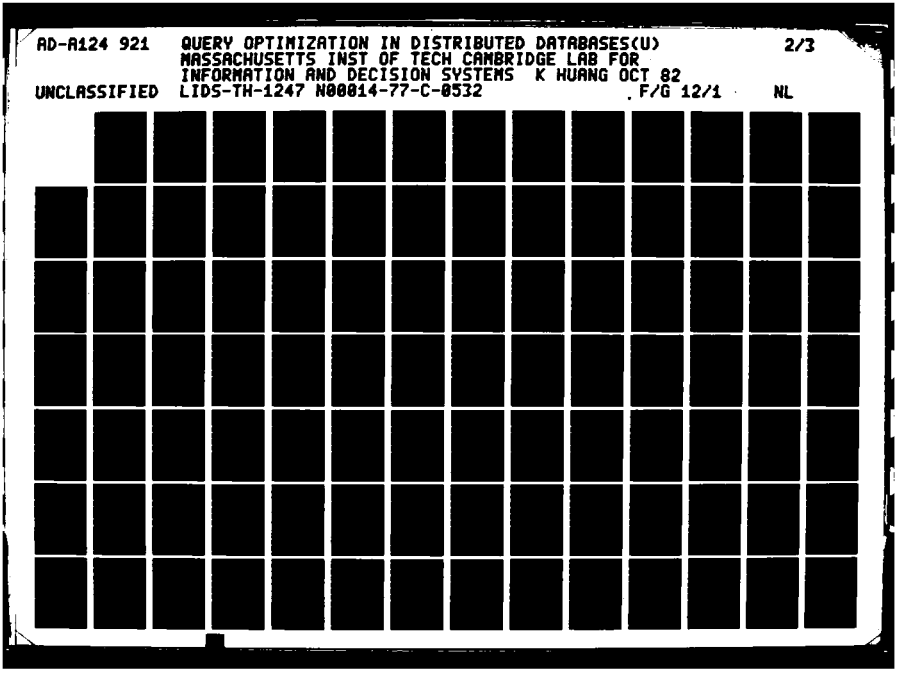
$$+ (S_1 \cdot S_2 \cdot S_3 \cdot d_{13} \cdot d_{23} \cdot S_4 \cdot d_{34}) \cdot C_{40}$$

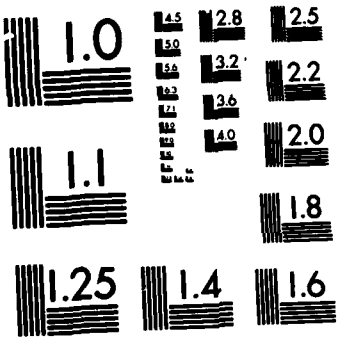
FIGURE 3.1 THE COST OF A SUBTREE

the join operation between them will produce the size of data at node j equal to $s_i\ s_j\ d_{ij}$. Here, $d_{ij} = d_{ji}$ for each pair of i,j . We define a subtree of a tree as the usual meaning of subtree with all edges pointing to the direction of the root of the subtree. Figure 3.1 illustrates an example of the cost of a subtree.

Our objective is to find an inversely directed spanning tree toward node V_0 with minimum communication costs. Note that this model is different from the model stated in chapter two. The model in this chapter is not consistent in estimating the sizes of the data by two strategies which will generate the same results. For example, in figure 3.1, If we join data in node 1, 2 and 3 by joining data from node 1 to node 3 and then joining data from node 2 to node 3 will result the size be $s_1*s_2*s_3*d_{13}*d_{23}$. On the other hand, if we do it by joining data from node 1 to node 2 and then joining the result from node 2 to node 3 will result the size be $s_1*s_2*s_3*d_{12}*d_{23}$. We will show that under this model, three problems of finding a routing strategy of sending required data to the desired site are NP-complete.

This chapter is organized as follows: In the next section, we review complexity theory and list the three NP-complete problems we need to use for proving our results. In section 3.3 , we prove the NP-complete results for the query processing problem with a bounded number of nodes in each subtree (QPBS) by using the satisfiability problem


AD-A124 921 QUERY OPTIMIZATION IN DISTRIBUTED DATABASES(U) 2/3
MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR
INFORMATION AND DECISION SYSTEMS K HUANG OCT 82
UNCLASSIFIED LIDS-TH-1247 N00014-77-C-0532 F/G 12/1 NL



(SAT), for the query processing with a bounded number of subtrees (QPBT) by using the bounded component spanning forest problem (BC), and for the query processing problem with maximum benefit (QM) by using the exact cover by 3-sets problem (X3C). The conclusion is given in section 3.4.

3.2 Complexity Theory

Recent developments in the theory of computational complexity provide a powerful method for comparing the computational difficulties of different problems. It often can provide information useful to algorithm designers. The application of this theory to combinational problems has aroused the interest of many researchers. The foundations for the theory are in the paper of Cook [Cook 71] and of Karp [Karp 72] who first explored the relation between the classes P and NP of (language recognition) problems solvable by deterministic and non-deterministic turing machines, respectively, in a number of steps bounded by a polynomial in the length of the input. In this context, all problems are stated in terms of recognition problems which require yes/no answers. For the combinatorial optimization problem, we transform it into the problem of determining the existence of a solution with value at most (or at least) equal to y, for some threshold y. The class NP is very extensive. It contains several classical problems ranging from the satisfiability problem of propositional calculus to the traveling salesman problem for which, despite many

lengthy and intensive efforts, no efficient algorithm is known at present.

A problem is said to be NP-complete, if, intuitively, it is as hard as any problem in NP. Proving that a given problem is NP-complete typically requires two steps:

1. Showing that this problem is in NP by describing an efficient nondeterministic algorithm solving it.
2. Showing how a known NP-complete problem can be reduced to the given problem via a polynomially time-bounded transformation.

The following theorem lists those NP-complete problems that will be used in this chapter to establish NP-completeness of the distributed query processing problem.

Theorem 1: The following problems are NP-complete:

(1) Satisfiability problem in the conjunctive normal form (SAT)

Given a set $\{x_1,\ldots,x_n\}$ of variables and a set $c_1,\ldots,c_m$ of clauses. Each clause is the disjunction of literals (i.e. variables or negations of variables). We are asked to determine whether or not the conjunction of $c_1,\ldots,c_m$ is satisfiable, i.e.,whether there is an assignment of the values true and false to each of the variables, so that each clause contains at least one true literal.

(2) Bounded component spanning forest (BC)

Given a complete graph G=(V,E), and given a nonnegative weight $W(v) \in Z_0^+$ associate with each $v \in V$.
Assume positive integers $K \leq |V|$ and B are also given. Can the vertices in V be partitioned into $k < K$ disjoint sets $V_1, V_2, \ldots, V_k$ so that for $1 \leq i \leq k$, the subgraph of G induced by V_i is connected and the sum of the weights of the vertices in V_i does not exceed B?

(3) Exact cover by 3-sets (X3C)

Given a finite set $X=\{\tau_1, \tau_2, \ldots, \tau_{3q}\}$ with $|X|=3q$ and a collection $C=\{\sigma_1, \sigma_2, \cdots, \sigma_\gamma\}$ of 3-element subsets of X . Does C contain an exact cover for X ? i.e., Does there exist a subcollection $C' \subseteq C$ so that every element of X occurs in exactly one member of C' ?

Proof:

(1) see reference [KARP 72]

(2) see reference [GAJO 79]

(3) see reference [KARP 72]

3.3 Computation Complexity of Query Processing Problem.

In this section, we show that three problems of finding a routing strategy of sending required data to the desired site are NP-complete. The first problem is the query processing problem with the constraint that the number of nodes in each subtree is bounded. The second problem is the

query processing problem with the constraint that the number of subtrees of the root of the directed spanning tree is bounded. The third problem consider the maximun benefit of routing strategy which has nothing to do with the minimum cost query processing problem.

3.3.1 Query Processing Problem With a Bounded Number of Nodes in Each Subtree (QPBS)

We formulate the problem as follows:

Assume we are given:

* a complete graph G=(V,E)
* s_i, the size of data associated with each node v_i
* c_{ij}, the unit transmission cost associated with edge (i,j)
* d_{ij}, the reducibility associated with each pair of nodes (i,j)
* B is rational constant and K is integer constant.
* v_0, the final node (query answer node)

Does there exist a directed spanning tree toward v_0 so that the cost of the spanning tree is at most B and the number of nodes in each subtree of v_0 is at most K?

Theorem 2: QPBS is NP-complete

Proof:

Consider the SAT problem in theorem 1. We will show that SAT is reducible to QPBS, i.e., that for any instance of SAT an instance of QPBS can be constructed in polynomial-bounded

time so that solving the instance of QPBS solves the instance of SAT as well. The theorem then follows from the NP-completeness of SAT in theorem 1 and the fact that QPBS belongs to NP, since any feasible spanning tree toward node v_0 can be recognized as such in polynomial time.

Given any instance of SAT, we write a set of variables as $\{x_1, \ldots, x_n\}$ and a set of clauses as $\{c_1, c_2, \ldots c_m\}$, and define an instance of QPBS as follows:

G=(V,E) is a complete graph such that:

* $V=\{v_0, x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n, c_1, \cdots, c_m, d_1, \cdots, d_m\}$
* each node is associated with a size 2, i.e. $s_i = 2 \ \forall i$
* each edge is associated with a reducibility = 1/2
* the cost of each edge is defined as follows:

  cost $(d_j, d_1) = 1 \quad \forall j = 2, \cdots, m$

  cost (d_1, x_1) = cost $(d_1, \bar{x}_1) = 1$

  cost (x_i, x_{i+1}) = cost $(x_i, \bar{x}_{i+1})$ = cost $(\bar{x}_i, x_{i+1})$ = cost $(\bar{x}_i, \bar{x}_{i+1}) = 1$

  $\forall i = 1, 2, \cdots, n-1.$

  cost $(x_n, v_0) = 1$, cost $(\bar{x}_n, v_0) = 1$

  cost $(c_i, \sigma_j) = 1$ if σ_j is a literal in c_i where σ_j is either x_j or $\bar{x}_j$.
* all other edges have high cost.
* B = 2 (2n+2m)
* K = n+m

Figure 3.2 illustrates this reduction.

SAT : THE BOOLEAN FORMULA $B = (\bar{x}_1 + \bar{x}_2) * (x_2 + x_3) * (x_1 + x_2 + \bar{x}_3)$

QPBS :

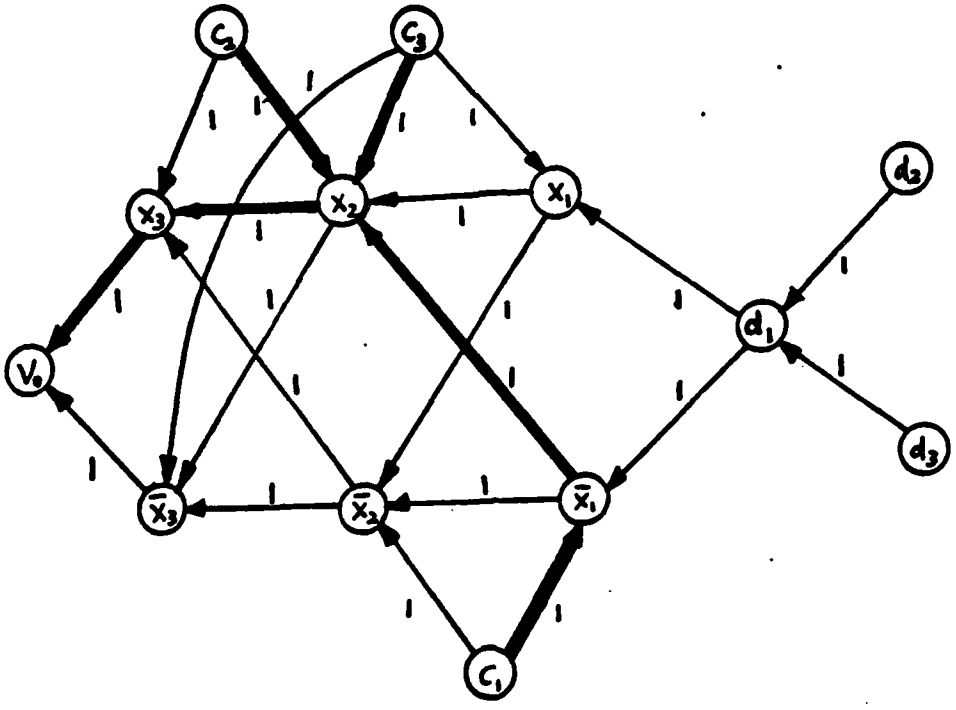


FIGURE 3.2 ILLUSTRATION OF REDUCTION FROM SAT TO QPBS

Each node is associated with a size 2 and each edge has a reducibility of 1/2 . Edges of infinite length are not shown. The solution of QPBS corresponding to the truth assignment x_1= false, x_2= x_3=true is shown with a heavy line.

We claim that SAT has a solution if and only if G =(V,E) contains a spanning tree towards v_0 so that the cost of each subtree is bounded by B and the number of nodes in each subtree of v_0 is bounded by K.

($\Rightarrow$) Assume there is a truth assignment t satisfying SAT. Then the tree T with d_2 ,d_3 ,..., $d_n$through d_1 and then through those 0-value literals of $\{x_i, \bar{x}_i\}$ and then to v_0 as one subtree, and c_1 ,c_2 ,..., c_m through the 1-value literals and then to v_0 as another subtree of v_0 will have cost 2 (2n+2m) and each subtree has n+m nodes.

($\Leftarrow$) Suppose QPBS has a tree solution T. Note that T must have 2n+2m edges. Since B =2*(2n+2m), and the shortest edge of G has length 1, it follows that T must consist of 2n+2m edges of length 1. Also, since there are two edges of length 1 incident upon node v_0 and K is half the number of nodes in G, T has exactly 2 subtrees rooted at node v_0 , each of cardinality (not including v_0) n+m. The m nodes d_1 ,d_2,..., d_m are obviously in one such subtree. The only way that these nodes can be connected to x_1 or $\bar{x}_1$ is by a path P traversing one node from each pair $\{x_i , \bar{x}_i\}$.

Let a literal have the value false if the corresponding node is in P. Since $d_1, d_2, \ldots, d_m$ together with P exhausts the m+n vertices allowed in a subtree, the nodes $\{x_i, \bar{x}_i\}$ not in P and $\{c_j\}$ must constitute the other branch. Also, for each c_j, there is a node not in P, such that c_j is adjacent to it. Hence each clause of SAT contains a literal that is assigned the value true by the above truth assignment. Thus SAT is satisfiable.

3.3.2 Query Processing With a Bounded Number of Subtrees (QPBT)

We next consider the objective to be finding a directed spanning tree toward final node v_0 with a minimum number of subtrees such that the cost of each subtree is bounded by B. We prove that this problem is still NP-complete.

This problem is formulated as follows:

Assume we are given

* G = (V,E) a complete graph
* s_i, the size of data associated with each v_i
* c_{ij}, the unit transmission cost associated with edge (i,j)
* d_{ij}, the reducibility associated with each pair of nodes (i,j)
* B and K, constants
* the final node, v_0

Does there exist a spanning tree toward node v_0 such that the cost of each subtree is bounded by B and the number of

the subtrees is no greater than K?

Theorem 3: QPBT is NP-complete

proof:

Note that the proof of theorem 2 works trivially for theorem 3 also. Here we give another proof. Consider the BC problem of Theorem 1. We will show that BC is reducible to QPBT, i.e., that for any instance of BC an instance of QPBT can be constructed in polynomial-bounded time so that solving the instance of QPBT solves the instance of BC as well. The theorem then follows from the NP-completeness of BC in theorem 1 and the fact that QPBT belongs to NP, since any feasible spanning tree toward node v_0 can be recognized as such in polynomial time.

Given any instance of BC, we denote the graph by $G_B = (V_B, E_B)$, and the weight of each node $v_i \in V_B$ by $W(v_i) \in Z_0^+$. We construct an instance of QPBT as follows:

* $G = (V,E)$ is a complete graph with $V = V_B \cup \{v_0 \text{ (final node)}\}$
* $s_i = 1 \quad \forall i$
* $c_{ij} = W(v_i) \quad \forall i$
* $d_{ij} = 1 \quad \forall i,j$
* K & B are the same as in BC

The subtree example following has the communication cost $W(v_1) + W(v_2) + W(v_3) + W(v_4)$.

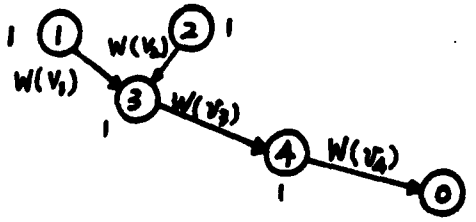


FIGURE 3.3 ILLUSTRATION OF REDUCTION FROM BC TO QPBT

We claim that BC has a solution iff the constructed G =(V,E) contains a spanning tree toward v so that the cost of each subtree is bounded by B and the number of the subtrees is bound by K.

($\Rightarrow$) Assume that BC has a solution
i.e. the vertices in V can be partitioned into $k \leq K$ disjoint sets $V_1, V_2, \ldots, V_k$ so that for $1 \leq i \leq k$, the subgraph of G induced by V_i is connected, and the sum of the weights of the vertices in V_i is $\leq B$.
We construct a subtree T_i for each set V_i and connect it to v and each subtree T_i has comm. cost = $\sum_{l \in T_i} s_l = \sum_{l \in T_i} W(v_l) \leq$ B and the number of subtrees $k \leq K$.

($\Leftarrow$) The reverse of the argument is the same as above.

3.3.3 Query Processing Problem With Maximum Benefit (QM)

In the query processing problem, if we associate each edge with the unit benefit of transmitting data through that edge instead of with the unit cost, then the problem becomes finding a maximum benefit spanning tree toward node v_p. We show that this problem is still NP-complete. Note that this problem has nothing to do with minimum cost query processing problem because we cannot transform minimum cost QP problem to this problem.

We formulate the problem as follows:

Assume we are given:

* G = (V,E)
* for each node v_i, associate data size, s_i
* for each pair of nodes (i,j), associate a join reducibility d_{ij}, and a unit benefit b_{ij}
* final node v_o
* constant B

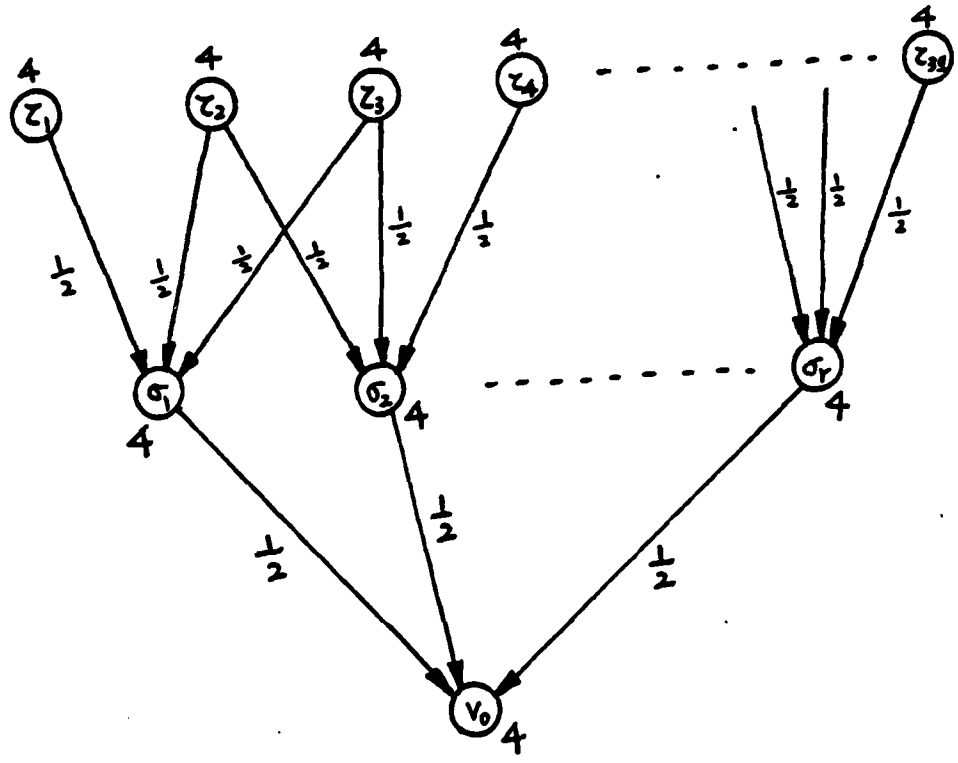
Does there exist a spanning tree toward v_o such that the total benefit is no less than B ($\geq$ B)?

Theorem 4: QM is NP-complete

proof:
Considering the X3C problem in theorem 1. We will show that x3c is reducible to QM, i.e., that for any instance of X3C, an instance of QM can be constructed in polynomial-bound time such that solving the instance of QM solves the instance of X3C as well. The theorem then follows from the NP-completeness of X3C in theorem 1 and the fact that QM belongs to NP, since any feasible spanning tree toward node v can be recognized as such in polynomial time.

Given any instance of X3C, we write a finite set of elements $X = \{ \zeta_1, \zeta_2, \ldots, \zeta_{3q} \}$ and a collection C of 3-element subsets of X: $C = \{ \sigma_1, \sigma_2, \ldots, \sigma_r \}$.

We define an instance of QM as follows:



EDGES OF $-\infty$ BENEFIT ARE NOT SHOWN

FIGURE 3.4 ILLUSTRATION OF REDUCTION FROM X3C TO QM

G =(V,E) is a complete graph

* V = { σ_1 , σ_2 ,..., σ_r , z_1 , z_2 ,..., z_{3q} , v_0 }
* for each node i, s_i =4
* d_{ij} = 1/2 $\forall$ i,j
* b (z_i , σ_j) = 1 if $z_i \in \sigma_j$
  b (σ_j , v_0) = 1 j = 1,2,...,r
  b (m, n)=-∞ otherwise
* B = 40q + 4r

Figure 3.4 illustrates this reduction.

We claim that X3C has a solution if and only if G =(V,E) contains a spanning tree with the total benefit no less than B.

($\Rightarrow$) X3C has an exact cover c'$\subseteq$C, then the spanning tree with edges (z_1 , σ_j) $\forall \sigma_j \in C'$ and (σ_i , v_0) i= 1,2,..r, has benefit

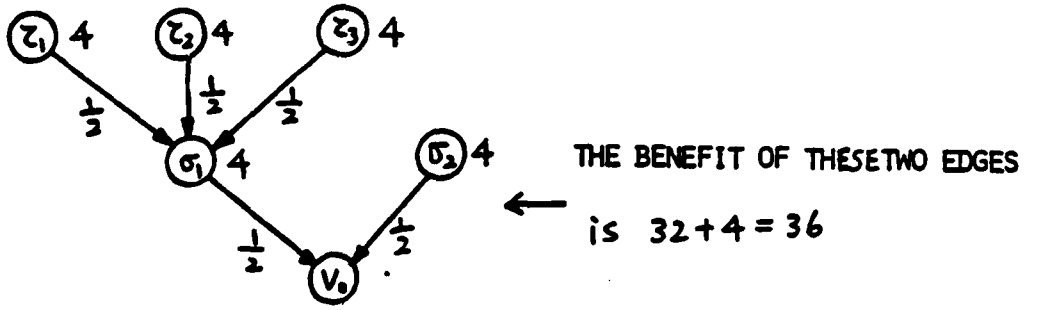
3q 4 + 32 q + 4 (r-q)

=44q - 4q + 4r

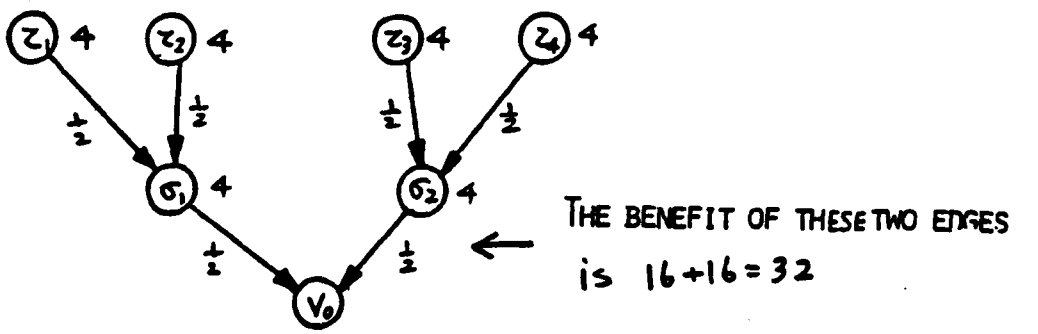
=40q + 4r > B

($\Leftarrow$) Considering the spanning tree toward node v_0 in fig. 3.4, there are only three types of first-level nodes directed toward the second-level nodes: (1) the nodes in the first level direct toward one node in the second level and leave one node in the second level without any first-level nodes directed to it, (2) two second-level nodes each have two first-level nodes directed to them, and (3) one

T_1: The benefit of T_1 is $12+32+4=48$



T_2: The benefit of T_2 is $16+16+16=48$



T_3: The benefit of T_3 is $12+16+8=36$

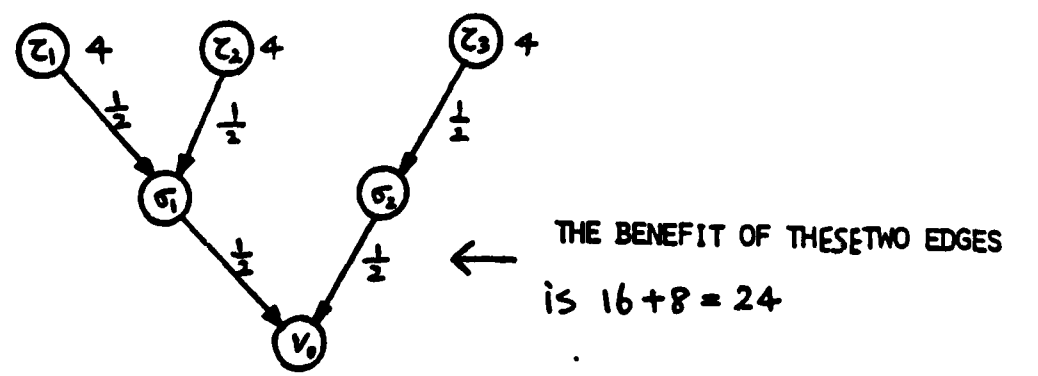


FIGURE 3.5 THREE TYPES OF SPANNING TREES TOWARD NODE V_0

second-level node has two first-level node directed to it and the other second level node has only one first-level node directed to it. We illustrate the three cases as in T_1, T_2, and T_3 respectively in figure 3.5.

As we can see from figure 3.4 and figure 3.5, we know

* The benefit of the first level in figure 3.4 is constant 12q.
* The maximun benefit of the second level in figure 3.4 is 32q + 4 (r - q) which occur when exactly three z_i connect to one σ_j .

So if there exists a spanning tree toward v_0 with cost no less than B ($\geq$ 40q + 4r), then this must be a spanning tree with exact three z connect to one σ_j .
Let

$$C' = \{ \sigma_j \mid \text{exact three } z_i \text{ connect to one } \sigma_j \text{ in the spanning tree} \}$$

Then, C' is the exact cover of X3C.

Following the same proof, if we restrict $d_{ij} = 1/2$ for all i, j then the problem is still NP-complete.

3.4 Conclusion:

We have proven that three variations of the query processing problem (QP) are NP-complete problems. The complexity of the query processing problem is still open. It

is a future research problem. The difficulty of solving query processing problems comes from the fact that the cost of each stage depends on the cost of all previous stages. That is, we need to search over all the inversely directed spanning trees toward the final node v and compute the cost of them. The solution space is very large. The other complexity comes from the fact that at each stage (node), we want to choose the next node to move and process; we have two objectives: one is the minimization of the immediate transmission cost and the other is the maximization of the data being reduced after moving the data from one node to the other and performing the join operation between them.

Because of the results of this chapter and the dynamic programming problem nature of the query processing problem, we do not intend to look for optimal algorithms. Instead, our objective in the next chapter is to provide heuristic algorithms for choosing routing strategies for processing a given query.

Future research direction will be determing the complexity of query processing prblems in the model of chapter two.

Chapter 4

Heuristic Algorithm for Distributed Query Processing Problem

4.1 Introduction

In the last chapter, we have shown that under three different objective functions, the problem of finding a routing strategy for sending required data to the site where a query is initiated is NP-complete. The exact algorithm for this problem must enumerate all possible solutions and calculate their costs because of their dynamic network nature. When the number of nodes is small, the enumeration algorithm is possible and hopefully can generate an answer within a reasonable response time. Since many of the queries issued by users are for real-time application, response time is an important factor for the users. The algorithm for a generating routing strategy must be very efficient in order to meet the requirement. Heuristic algorithms appear to be the only reasonable option for solving such problems.

In this chapter, we will analyze the problem and provide heuristic algorithms for the problem. We first consider the simpler case of the problem where all possible semijoins are performed and only consider the routing strategies for join operations. We provide several heuristic algorithms for this problem. We then extend the algorithms of the simpler case to the general distributed query processing problem of solving an equi-join query by a

sequence of join-semijoin mixed operations. Numerical examples are given to show how the algorithm work.

Before we discuss algorithms, we shall introduce several graphical terminologies as used in graph theory. An inversely-directed spanning tree is a directed spanning tree with all edges pointing in the direction of the root of the tree. A branch of a node is a subtree of that node. Consider a directed path (x_1,x_2) (x_2,x_3) .... (x_{k-1},x_k) passing through nodes $x_1,x_2,\ldots,x_k$ in a graph. The length of this path is k, i.e., the number of nodes in the path. For i and j so that $1\leq i<j\leq k$, x_j is a successor of x_i and x_i is a predecessor of x_j. If $j=i+1$, we shall use the terms immediate successor and immediate predecessor, respectively. A node with no successor is a terminal node (or final node), and one with no predecessors is an initial node. A nonterminal node is at level k in a graph if the longest path from it to a terminal node is of length k. The level of a terminal node is defined to be 1. A sequence of join-semijoin operations is a sequence of join-semijoin edges in the query processing graph which we defined in chapter 2.

4.2 Cases Where All Semijoin Reducibilities Equal Zero

In this section, we consider a simpler case which assumes that all possible semijoins were performed, i.e., all semijoin reducibilities between each pair of nodes

become zero. We also assume that the join reducibilities are not changed or affected by the join operation. This assumption simplifies the problem by ignoring the possible dynamic changes of join reducibilities after join operation.

We describe the problem in the following: [SQP]

Given a complete directed graph G=(V,E) with V={ v_0 ,v_1 ,...,v_n } and E={ (v_i ,v_j)| V i ≠ j }. Assume v_0 is the answering node of a query. For each node v_i, we denote the data size of that node by s_i. With each pair of nodes (v_i ,v_j), we associate a unit transmission cost c_{ij} of that edge and a reducibility d_{ij} between relations in node v_i and v_j . The reduciblity d_{ij} between relations in nodes v_i and v_j means that if we send a relation from node v_i to node v_j and perform a join operation locally at node v_j, the resulting data at node v_j will have size $s_i \cdot s_j \cdot d_{ij}$. In fact, this d_{ij} has the same interpretation as $1-\beta_{ij}$ where β_{ij} is the join reducibility between relations in node v_i and v_j . Figure 4.1 gives an example of this problem. The total transmission cost of the solution strategy is computed by taking the sum of the subtree cost of each branch of the tree. Figure 4.2 shows an example of the transmission cost of the operation between a pair of nodes. The costs are labelled next to the edges. Our objective is to find a directed spanning tree toward node V_0 with minimum total communication cost.

4.2.1 Analysis

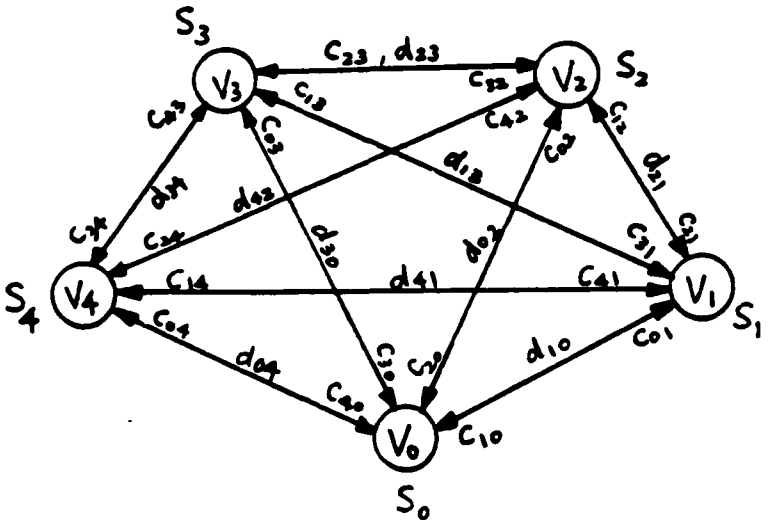
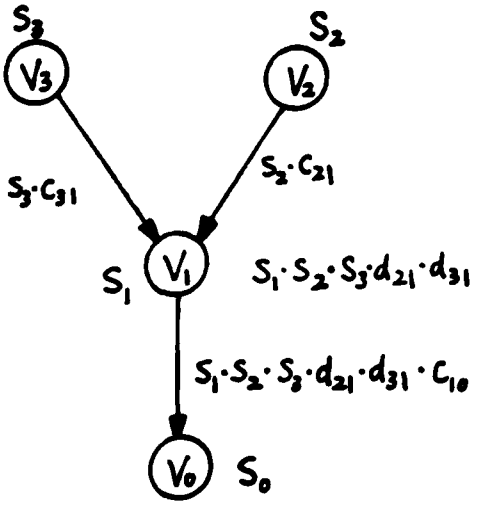


FIGURE 4.1 EXAMPLE OF SQP PROBLEM



TOTAL COST OF THIS SUBTREE IS

$$S_3 \cdot C_{31} + C_2 \cdot C_{21} + S_1 \cdot S_2 \cdot S_3 \cdot d_{21} \cdot d_{31} \cdot C_{10}$$

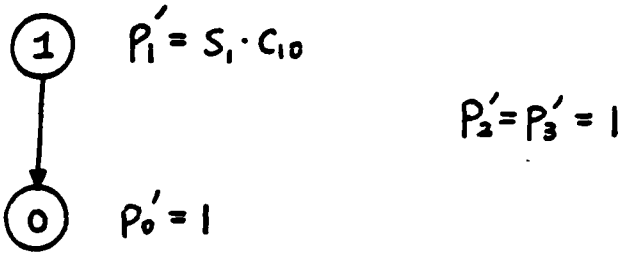
FIGURE 4.2 EXAMPLE OF COMPUTING SUBTREE COST

In this problem, we have two types of objectives at each step. The first type is to minimize the next step transmission cost, i.e. node i, $\min_j \{c_{ij}\}$. The second type concerns data reduction after this operation. It can be either finding the minimum of the data required to transmit after the first-step operation, i.e. $\min_j \{d_{ij} \cdot s_j\}$ or finding the maximum of the data being reduced after the first-step operation, i.e. $\max_j \{(1-d_{ij}) \cdot s_j\}$. In order to find the next step operation, we either can consider only one of the two types of objectives or consider the weighted combination of those two objectives. This type of heuristic algorithms forms a solution in a single pass by selecting operations sequentially in an order that minimizes the increase in the objective at each step. We call this type of algorithm a greedy heuristic because of its appetite for immediate improvement. We will give an analysis of this problem in this section in order to provide a constructive insight of the heuristic algorithm.

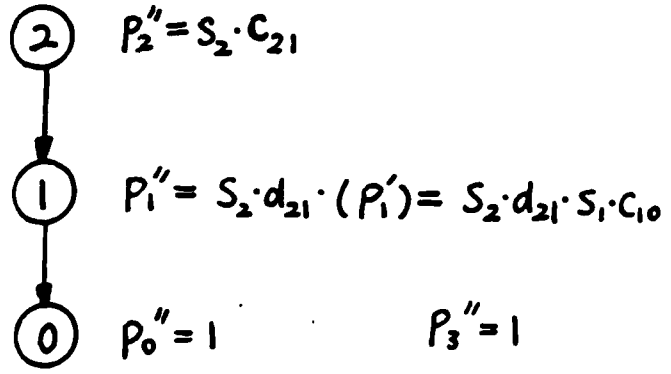
We define a node label p_i for each node to be the transmission cost of the next edge in the solution tree. Initially, we assume all node labels $p_i = 1$. As the solution tree grows, the number of nodes in the tree increases and the node labels will update accordingly. Fig. 4.3 gives an example of updating node labels of the example in fig. 4.2

A. Initially $P_0 = P_1 = P_2 = P_3 = 1$

B.



C.



D.

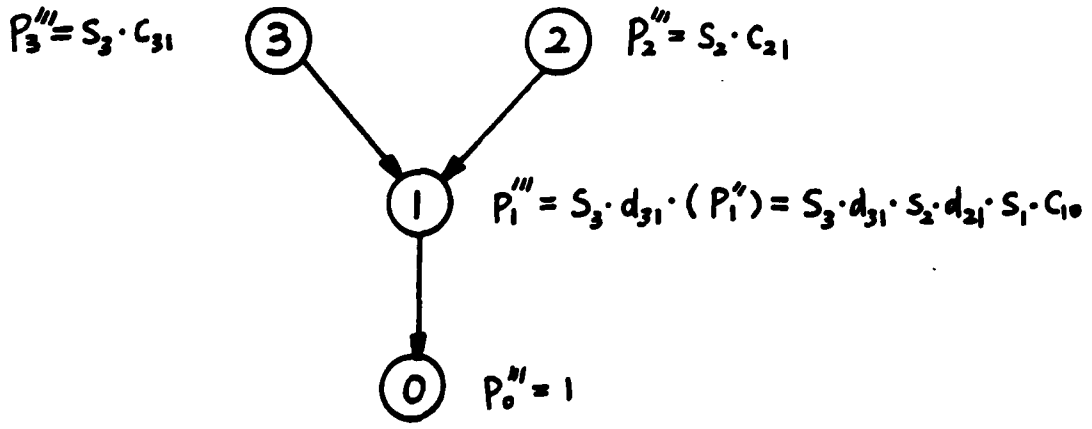


FIGURE 4.3 EXAMPLE OF UPDATING NODE LABELS

At any stage, if we want a new node to be included in the solution tree, we need to compare the total transmission costs of all possible cases. For example in fig. 4.3, if node 4 is the next node to be included in the solution tree, it can be sent to node 0,1,2, or 3. We need to compare all four possible cases and choose the one with minimum total communication costs. See fig. 4.4. In general, if S is the set of nodes in the solution tree at this stage and node k is the node which will be included in the solution tree, we need to compare the costs

$$\min_{i \in S} \{ s_k * c_{ki} + s_k * d_{ki} (\sum_{\substack{j \text{ in path}(i,0) \\ j \neq 0}} p_j) + \sum_{\substack{j \text{ not in path}(i,0) \\ j \neq 0}} p_j \} \qquad (4.1)$$

and choose a minimum one. Assuming node r to be such a node, we include node k in the solution tree by adding the edge (k,r). The node labels are changed as follows:

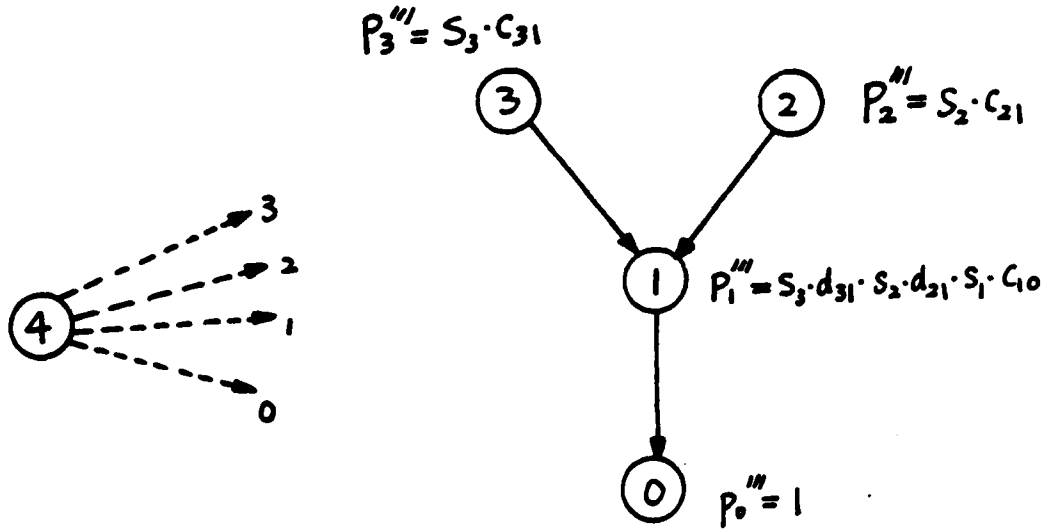
$p'_k = s_k * c_{kr}$

$p'_j = s_k * d_{kr}(p_j)$ $\quad \forall$ j in path (r,0) & j ≠ 0.

$p'_i = p_i$ otherwise.

Figure 4.5 gives the node label's updating of Figure 4.4 where node 3 is the node which was selected.

Assuming the graph has node V_0 and n other nodes, and the number of nodes in the solution tree S at this stage is m, the number of operations at each stage is then $O(n \cdot m)$.



$$\min \begin{cases} S_4 \cdot C_{43} + S_4 \cdot d_{43} (P_3''' + P_1''') + P_2''' \\ S_4 \cdot C_{42} + S_4 \cdot d_{42} (P_2''' + P_1''') + P_3''' \\ S_4 \cdot C_{41} + S_4 \cdot d_{41} (P_1''') + P_2''' + P_3''' \\ S_4 \cdot C_{40} + P_2''' + P_3''' + P_1''' \end{cases}$$

FIGURE 4.4 INCLUDING A NEW NODE INTO SOLUTION TREE

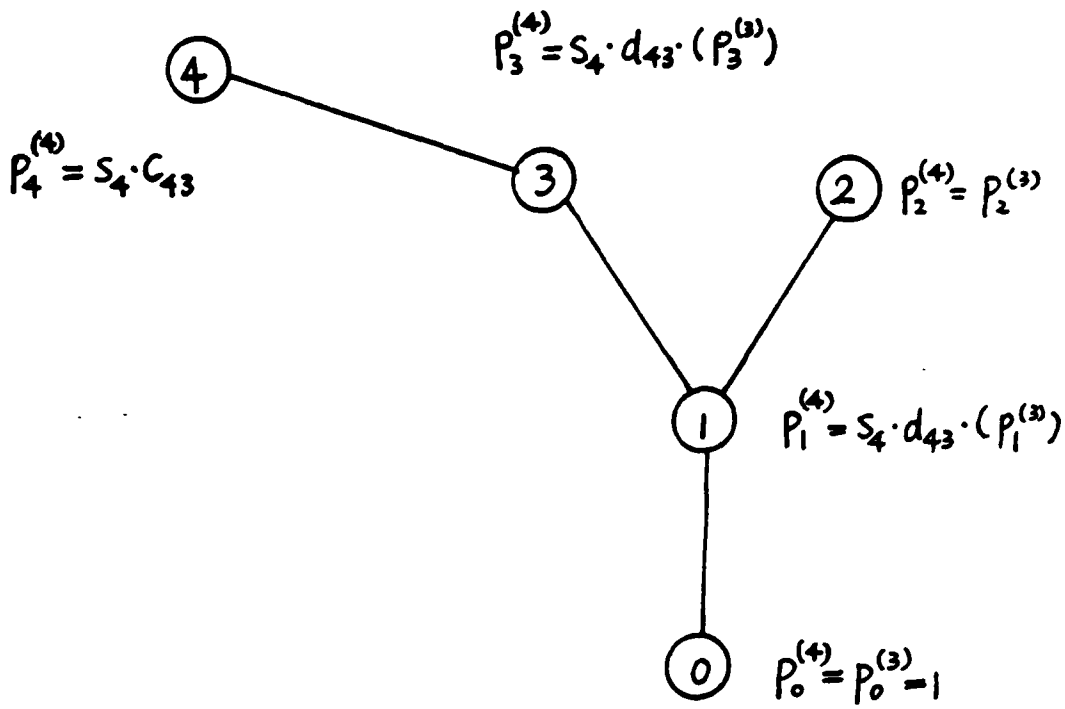


FIGURE 4.5 UPDATING NODE LABELS

So the number of computations of this greedy algorithm requires $0(n^3)$. We will identify several special cases in which the computations are reduced.

Let $I=\{i \mid C_{i0} = \min_j\{C_{ij}\}\}$ be the set of nodes where the minimum unit transmission cost is desired when sending the query answering node 0, and $J = \{i \mid \min_{j\neq 0} \{s_i \cdot d_{ij}\} < 1\}$ be the set of nodes where there exists a non-final node at which data can be reduced to a smaller size by sending the data from node i and performing the local processing.

Given a node i, we can classify into one of the following four cases:

	$i \in I$	$i \notin I$
$i \in J$	①	②
$i \notin J$	③	④

Case 1: $C_{i0} = \min_j \{C_{ij}\}$ and $\exists k$ so that $s_i \cdot d_{ik} = \min_{j\neq 0} \{s_i \cdot d_{ij}\} < 1$

Since the node labels will change as $P_j' = s_i \cdot d_{ik}$ (P_j) $\forall$ j in path (k,0), $j \neq 0$, if we add the edges (i,k), and $s_i d_{ik} < 1$, it will reduce the

total communication cost of the path (k,0). We need to compare the tradeoff between the payment for the first step transmission cost and the gain from the saving of the path transmission cost.

The strategy of choosing k is such that k is the outermost node of a branch with larger $\sum_{j \in path(k,0)} P_j$ and $(1 - s_i \cdot d_{ik} \cdot \sum_{j \in path(k,0)} P_j).br \geq s_i(c_{ik} - c_{i0})$.

Case 2: $i \notin I$ and $i \in J$.

Let $K_i = \{k \mid C_{ik} < C_{i0}$ and $s_i d_{ik} < 1\}$.

Lemma: If $K_i \neq \phi$, then we never use edge (i,0)

Proof:

Since $s_i \cdot d_{ik} < 1$ implies

$s_i \cdot d_{ik} (\sum_{j \in path(k,0)} P_j) \leq \sum_{j \in path(k,0)} P_j$

and $s_i c_{ik} \leq s_i c_{i0}$, the cost of sending data from node i to node 0 will have a higher communication cost.

If $K_i = \phi$, then we must compare all options as in the general case.

Case 3: $i \in I$ and $i \notin J$

Since $i \in I$, so $s_i * c_{ij} > s_i * c_{i0}$, $\forall j$, and $i \notin j$ implies $s_i * d_{ij} > 1$. The cost of sending data directly from i to 0 will be $s_i * c_{i0}$. And the total cost

will be

$$s_i c_{i0} + \sum_{\substack{\text{all nodes } j \\ j \neq 0}} p_j \quad . \quad (4.3)$$

The cost of sending data to node k will be $s_i \cdot c_{ik}$ and the total cost will be

$$s_i \cdot c_{ik} + s_i \cdot d_{ik} \left(\sum_{j \in \text{path}(k,0)} p_j \right) + \sum_{\substack{\text{all other nodes } j \\ j \neq 0}} p_j \quad (4.4)$$

Formula 4.4 is always greater then (4.3), so we have :

Lemma: If $i \in I$ and $i \notin J$, then the minimum routing is through edge (i,0).

Case 4: $i \in I$ and $i \notin J$.

In this case, we need to compare all possible options to include node i in the solution tree. We use formula (4.1).

4.2.2 Heuristic Algorithms

In general, a heuristic algorithm can be divided into two stages. The first stage is to build a feasible solution. If the algorithm for generating an initial feasible solution is good enough to provide a solution close to the optimal solution, then the algorithm itself can be used as a heuristic for the problem. The second stage is the improvement stage. At this stage, the heuristic attempts to improve an initial feasible solution by some method to obtain a better solution. The most often used

method is the interchangeable heuristic which attempts to improve an initial feasible solution by interchanging some nodes in the solution with some that are not in the solution. This process continues until a solution is found that cannot be improved by such interchanges.

In this section, we will seperately describe the algorithms for building an initial feasible solution and for solution improvement. Any combination of algorithms for those two stages will be an algorithm for [SQP].

We will first describe the algorithms for building an initial feasible solution.

According to the analysis of section 4.2.1, we can easily generate an algorithm to build an initially feasible solution that is an inversely directed spanning tree toward node v , gradually letting T be the solution tree. Initially T is empty. We also assume each node i has associated with it a node label p . Initially $P = 1 \; \forall i$. The algorithm is as follows:

Algorithm A:

(1) $T = \phi$ initially

(2) $\forall$ node $t \in V = \{ v \mid c_{vo} = \min_j \{c_{vj}\} \; \& \; s_j d_{jk} \geq 1 \; \forall j \}$, includ (t,0) into the tree. i.e. $T = \{ (t,0) \mid \forall t \in V \}$

(3) Sort the remaining nodes in the increasing order of the size of the data at each node. Assume that the

order is $s_{\gamma(1)} \geq s_{\gamma(2)} \geq \ldots \geq s_{\gamma(8)}$.

(4) Add the nodes in the tree T according to the order of this sequence

(5) In adding node $v_{\gamma(i)}$ into the tree T, choose a node t already in the tree so that adding the edge $(v_{\gamma(i)}, t)$ gives a minimum cost increase.

(6) update node labels according to formula (4.2)

(7) stop when all nodes are included in the tree.

This algorithm is a greedy heuristic algorithm because t forms a solution by selecting an edge which minimizes the increase in the objective at each step.

If we consider only one objective, the first-step transmission cost, then we can transfer this problem into a minimum-directed spanning tree problem.

Algorithm B:

(1) Build a complete directed graph with one node corresponding to one database site.

(2) Associate each edge with the cost of the first step of the transmission. i.e. $s_i \cdot c_{ij}$ for edge (i,j)

(3) Find a minimum-directed spanning tree toward node 0.

Since there exists a polynomial time algorithm for the minimum directed spanning tree problem, this solution can be solved easily. Since this algorithm considers only one objective and ignores the other objective, reducing the

resulting data after this operation, the worst case of this algorithm could be very bad. So we expect that this algorithm must be used with an improvement method to provide an algorithm for [SQP].

Another algorithm is obtained by considering the objective of reducing data by intermediate processing only, and ignordinge the first-step transmission cost. As with algorithm B, we expect that this algorithm must be used with an improvement method to provide an heuristic algorithm for [SQP]. We describe the algorithm as follows:

Algorithm C:

(1) Build a complete graph with one node corresponding to one database site
(2) Associate each edge (i,j) with the amount of data being reduced, $s_i \cdot s_j(1-d_{ij})$
(3) Find a maximum matching of the graph
(4) Combine the two nodes in a matching as one and build another complete graph in which each edge is associated with the new amount of data between this pair of nodes, and find a maximum matching again.
(5) Repeat this procedure until the number of nodes in each branch of the tree is less than a fixed number.

Another algorithm is to take a weighted combination of these two factors. If we let $W_i = \sum_{j \neq i} d_{ij} / (n-1)$, which is the average unit transmission cost from node i to the other node, then we can associate each edge (i,j) with a combination of first-step transmission cost and weighted second-step transmission cost, i.e., weighted data reducing factor. This approach is expected to generate a better solution than algorithm B and algorithm C.

Algorithm D:

(1) Build a completed directed graph with one node corresponding to one database site.

(2) Associate each edge with the combination of the first-step transmission cost and the weighted data reducing factor of intermediate processing. i.e.

$$s_i \cdot c_{ij} + s_i \cdot d_{ij} \cdot s_j \cdot w_i \text{ for edge } (i,j).$$

(3) Find a minimum directed spanning tree toward node 0.

The algorithm for solution improvement is called the interchange heuristic. This method starts from one approximate solution and then perturbs it somewhat to see if a better solution results. If a better solution does result, the original solution is discarded and perturbations on the new solution are tried. Methods of this kind for the traveling salesman problem are described in [LIN 65] and [RSL 77]. We generalize these techniques as a

perturbation method in our heuristic algorithms.

Define a k-change of a problem as the deletion of k edges and their replacement by other k edges so that another solution is obtained. We also define a solution as k-optimal if no k-change produces a better solution.

We describe the algorithms for solution improvement by k-interchange heuristics in the following.

Algorithm k-interchange:

(1) Obtain a initial feasible solution.
(2) Repeat apply k-change to the current solution until the k-optimal has reached.

The heuristic algorithm for solving SQP can be the combination of one of the algorithms A through D with algorithm k-interchange for some fixed k.

4.3 Heuristic Algorithm for Distributed Query Processing Problem

For the general query processing problem, we want to solving a query by a sequence of join and semijoin operations. Our heuristic algorithm is to generate a sequence of join operations for solving the query with minimun cost by a similar algorithm of algorithm A. This is an incremental method for building up the solution. Then we check each edge to determine whether there exists a semijoin

operation which will make the join operation have less transmission cost. If such a semijoin operation exists, then we replace the current solution strategy by adding this semijoin edge to it. We describe the algorithm as follows:

Algorithm E:

(1) $T = \phi$ initially

(2) $\forall$ node $t \in V = \{v \mid c_{v_0} = \min_j \{c_{v_j}\} \ \& \ s_j \cdot d_{v_j} \geq 1 \ \forall j \}$, includ $(t,0)$ into the tree. i.e. $T = \{ (t,0) \mid \forall t \in V \}$

(3) Sort the remaining nodes in the increasing order of the size of the data at each node. Assume that the order is $s_{\gamma(1)} > s_{\gamma(2)} \geq \ldots > s_{\gamma(\ell)}$, where $s_{\gamma(i)} = r_{\gamma(i)} *$ Width of the relation.

(4) Add the nodes in the tree T according to the order of this sequence

(5) In adding node $v_{\gamma(i)}$ into the tree T, choose a node t already in the tree so that adding the edge $(v_{\gamma(i)}, t)$ gives a minimum cost increase.

(6) update node labels according to formula (4.2)

(7) After all nodes are included in the tree, we check each node to determine whether there exists one semijoin edge which will reduce the original transmission cost. If there exist one, then include the semijoin edge. The order of the checking sequence is from the terminal node up to its predecessor.

(8) Stop until all nodes have been checked for semijoin

operations.

We expect to have a numerical example for this algorithm. We also hope to generate other efficient algorithms.

4.4 Conclusions

In this chapter, we analyzed the difficult nature of the query processing problem and provided an analytical basis for heuristic algorithms. We first considered the simpler case of the problem where all possible semijoins are performed first, i.e., all semijoin reducibilities become zero. We provide several heuristic algorithms for this problem. Each of the algorithms has two stages. The first stage is to find a feasible processing strategy. The second stage is the improvement stage where interchange procedures are used. We then extend those heuristic algorithms to the general query processing problem by including semijoin operations into the sequence of join operations. We first create a solution strategy using only join operations and then change a join operation to one semijoin operation and one join operation when it is beneficial to do so.

Future research direction is the study of the analytic behavior of those heuristic algorithms. Although some analytic results of worst case and average case analysis are difficult to obtain, some computational experiments may be conducted to get a feeling for the average performance of

the algorithms. Developing more efficient and general solution procedures for general query processing problems is also a future research area.

Chapter 5

Query Processing in Distributed Heterogeneous Databases

5.1 Introduction

Database management systems are among the most important and successful software developments in this decade. They have already had a significant inpact in the field of data processing and information retrieval. The existing commercial systems are almost exclusively based on one of the three data models: relational, network(e.g., CODASYL) and hierarchical. Many organizations have independently developed their own databases on their own computers and database management systems to support the planning and decision making in operations. Each DBMS has its own intended schema, access control, degree of efficiency, security classification and operational requirements, etc. Often, different databases may contain data relevant to the problem although their structure and representation could be different. It will be beneficial if we can bring together all these databases in several locations in order to integrate information resources and build new kinds of applications to help operations.

Heterogeneous database management systems which are geographically distributed around the world play one of the most important roles in command, control and communication (C^3) systems, and other organizational operations. One of

the main problems in using these databases is the communication between them when we need to retrieve and update information. When a user issues a query at one site, the system must be able to respond to the user with an answer of the query as soon as possible. In today's business operations, efficient decision making based on information resources will depend increasingly on a more automated and faster response distributed database management system.

Existing data communication technology for computer networks does not yet provide a solution for the communication between these DBMSs. Communication delay of data transmission is still a dominant factor in system performance. In order to develop a method for query processing in distributed heterogeneous database management sytems environments and to develop a quantitative and syntactic understanding of the query processing strategies optimization, we need to have an integrated system to combine and share information in a heterogeneous distributed database environment.

In previous studies, a number of approaches have been proposed to the problem of heterogeneous DBMSs. One early proposal was to restructure each database into a common structure under a given DBMS; that is, to convert and to migrate the entire database from the various DBMS's to the given DBMS. This type of approach is generally called "data base conversion." [SHL 75], [SHTGL 77], [SLH 76], [SO 75],

and [SU 76] are works in this direction. Another approach is to maintain the original database and provide an effective information exchange among the different systems without incurring mass data migration. The major advantages and functional characteristics of this latter approach are:

1. A global data model is used to provide users with a common schema.
2. The data bases can be kept distributed without requiring any data movement.
3. Each data base can both operate in its own local mode and participate in the distributed system.
4. The application programs for the original data bases do not have to be changed and remain still usable.

The following researches are in this direction: In [AD 77] and [AD 78], the model of Abrial, [AB 74], is used as a global data model. In [NBC 76] and [CP 80], the Entity relationship (E-R) model proposed by [CHE 76] is used as global conceptual data model and a modified version of DIAM (Data independent accessing model) incorporating the syntax proposed by the CODASYL SDDTG (Storage Data Definition and Translation Group) is used as the global internal model [SDD 77]. In [SBD 81], MULTIBASE uses the Functional Data Model of [SHI 79] as the global conceptual data model. DAPLEX, embedded in the programming language ADA, is used as the user-interface language and a subset of ADAPLEX is used as a mapping language. The Database Communication System we propose in this chapter can be classified into the second

category. The main differences of our approach from others are that we take advantage of the relational data model and use a very high-level non-procedural language for user interfaces.

This chapter is dedicated to the study of communication among nonintegrated, heterogeneous and distributed DBMSs. A concept of a data communication system is proposed to provide a way to integrate and share information in a heterogeneous database. The Database Communication System is a front-end software system of a DBMS. It presents to users an environment of a single system and allows them to access the data using a high level data manipulation language without requiring that the total database be physically integrated and controlled. The architecture of the database communication system proposed in this chapter is only a conceptual design. All three components, schema unit, query unit and control unit require detailed requirement specifications. It is not our intent in this thesis to study all of them. We emphasize only the query unit in this thesis. In the next chapter, we will study how a query is processed in a heterogeneous database environment. Schema translation and query translation will also be addressed there.

The organization of this chapter is as follows: In section 5.2, we describe the motivations and difficulties of heterogeneous DBMS and specify the goal of this system

design. In section 5.3, we describe the components of heterogeneous database communication systems. The relational data model is then chosen as a global data model to support the communication. Several reasons are described in section 5.3.1. In section 5.3.2, we describe the architecture of a database communication system and the functional characteristics of each of its components. In section 5.3.3, some network configurations are described that permit integration of heterogeneous DBMSs by using database communication systems. In section 5.4, we described briefly how a query is processed in a heterogeneous database environment and we will leave the details of the query processing to the next chapter. Lastly, several other problems of database communication systems requiring further detailed specification are discussed.

5.2 Motivation and Objectives

5.2.1 The Heterogeneous World of DBMSs

In the real world, resources are heterogeous in nature, e.g. size, shape, color, structure etc. The same fact exists in the world of DMBSs. There are at least several dozens of heterogeneous DBMSs commercially available today, e.g., IMS,S2000, TOTAL IDMS, etc. From several points of view, we can distinguish heterogeous DMBSs.

1. Conceptual Model Viewpoint

Traditionally, the data model may be classified into three categories: hierarchical, network, and relational. Most of the commercially available systems may be implemented in some variant of one of the three models. For example , IMS is hierarchical, system 2000 is inverted hierarchical, TOTAL follows CODASYL DBTG architecture, ADABAS is inverted network and INGRES is relational.

2. Physical Model Viewpoint

Although two DBMSs may have the same conceptual model or may even be the same type of DBMS, they may have different data structures.

For example, the storing information about courses offered and courses taken by students may well use different physical data structures to represent it in a network model. With diferent data structures, the access paths will be different.

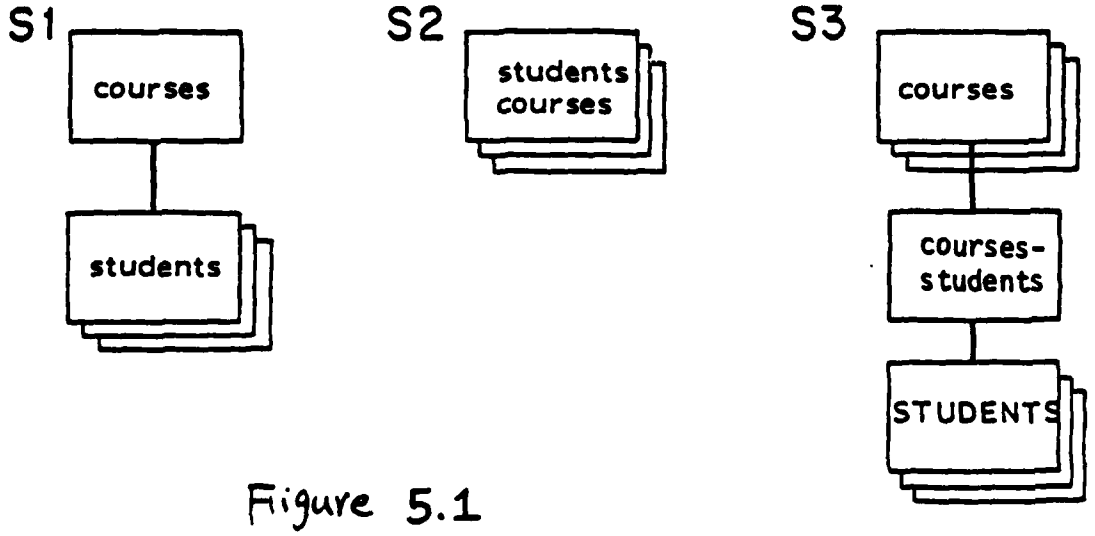


Figure 5.1

3. Data Manipulation Language Viewpoint

The data manipulation language can be record-at-a-time or set-at-time. In other words, it can be low-level procedural or high-level non-procedural. It depends on the conceptual model and physical model the system has adapted. It also depends on the system itself. For example, in the relational system, System R, the language can be SEQUEL or Query-by-Example.

4. Application Viewpoint

From an application point of view, the DBMS can be classified into a general-purpose system and a special purpose system. TOTAL is a general-purpose DBMS which is used for all kinds of different application purposes. The PARS (Programmed Airline Reservation) System is a special system which serves only a specialized application. The systems used for different purposes support different facilities.

5. Machine Viewpoint

The same DBMS can be implemented on different computers. The ARPANET-Datacomputer system is a typical heterogeneous system where quite different types of computers are tied together to implement their own DBMSs. Different computers may differ in their speed, memory size, storage management, etc.

6. System Control Viewpoint

Viewed from the system control aspect, there are two types of systems: centralized v.s. decentralized control systems. A centralized control system assumes the existance of one central control function to handle all systemwide global control. The LADDER-FAM (Language Access to Distributed Data with Error Recovery-File Access Manager) [1,2] developed at SRI is an example. A distributed control system where the control is completely distributed to each subsystem is more reliable. The SDD-1 system of the Computer Corporation of America [3] is an example of this type.

5.2.2 Difficulties and Approaches

The large bulk of local data are produced at a variety of locations in many fields. In business, scientific research and government, the data exchange is very important in decision making, experiment, management and control. The difficulties of communications between heterogeneous DBMSs can be identified as follows.

1. Data model --- the conceptual models for different DBMSs may be different. A user having a knowledge of one system may not be familiar with another system. Selection of a data model for every system to provide a uniform view to the end user is essential.

2. Data definition language --- in addition to selecting a data model, a data definition language to support the description of conceptual scheme is also essential.

3. Data manipulation language --- the user's query language cannot be the one for local host schemes; it must be a query language which supports the global uniform scheme. Because the end users don't know what data model the query will have to deal with, they are obviously unable to specify how something must be done, and so must instead specify what is to be done, i.e., the language must be nonprocedural.

4. Data integration --- most of the data base set up by independent organization are hard to integrate. It is also possible that inconsistencies exist between copies of the same information stored in different databases. Combining all local schema together to form a global schema is necessary in order to provide an integration schema for them.

5. Data incompatibilities --- the same objects in different DBMSs may be represented in different types, different schema names, different scales, etc. When integrating the DBMSs, we need to recognize these incompatibilities of data sources and identify them in the integration schema.

6. Processing results --- once a result is gotten for a query, it is expressed in the form of the original data model, and it must be translated to the uniform data model. Can this current result be saved and be operated on later?

7. Data dictionary and directory schema --- We must provide each end user with a uniform directory so that he is able to see easily what data is available, where it is, and how to get it.

8. Access planning --- with a high-level query language, the system should provide an optimizing strategy for each query in a distributed system.

9. Multiple-systems access --- each query may reference data in two or more different systems. The system must coordinate their transactions.

10. Multiple view support --- If the system wants to support multiple schemes for each DBMS, so that users can have freedom to choose their own preferred query language and global schema, then the systems must add more schema translators and query translators.

11. Control system --- After integrating different DBMSs, the system has to have a system controller to control the network DBMSs. The data manager must decide whether to use centralized control or distributed controls.

5.2.3 Design Objectives

Before we set up the design approach, it is important to decide what goals we want to achieve:

1. Central view for users --- All user's views are defined upon a global conceptual schema which is the union of the local schemata and integration schema. It is hoped that from the user's point of view, the system behaves in the same way as in a centralized system and the user is unaware of the fact that he may be dealing with heterogenous local databases.

2. General to any system --- We wish the database communication system to be general to any system and that it can be used to integrate various database systems for various applications. In addition, we want to minimize the cost and effort and maximize the performances.

3. Flexible to future extension --- We know that the volume and the complexity of database are extending very rapidly. It is the major factor of the cost of maintenance. We want the system to be flexible for the future extension with minimum cost.

4. Reliability --- The communications between heterogeneous DBMSs should not fully rely on a centralized system. The communication capability should be distributed among every heterogeneous DBMS.

5. Distributed control --- Based on the reliability and parallel processing issues, we want the communication between DBMSs to have distributed control.

6. Security --- When combining heterogeneous DBMSs, some confidential data in one system should often not be accessible so that access can be checked and the data protected.

5.3 Heterogeneous Database Communication Systems

5.3.1 Data Model

Because we are dealing with communications between different DBMSs supported by different data models,e.g. hierarchical, relational, etc., our approach is to select a data model to support a uniform conceptual schema for each DBMS. It provides users with a homogeneous view of conceptual schema and also serves as a bridge between the underlying models. Many logical data models have been proposed which model the real world in terms of the interested objects and the interrelation between them. In [KER 76], the authors study 23 data models and attempt to establish the similarities and differences among them according to data model structure, logical access type,semantics and terminology. Recent research has focused on two directions. One is to enhance the refinement of the conventional data models. The notion of "normal form theory" has led to a refinement of the relational model which

attempts to catch up more semantics information by explicitly expressing functional dependencies among data. Many authors worked along this direction and built some type of semanatic data models. The second approach has been to emphasize the identification of a basic simple construct. This construct is simple and with clean semantics. It may be easily collected in a meaningful fashion to represent complex varieties in semantic structures. It is clear that there is no mental model which is so superior that it is good for all users.

In view of the state of the art, we choose a relational data model as a global data model to provide a central view to the users' bases for the following reasons:

1. The relational data model shields the user from data formats, access methods and the complexity of storage structures.

2. It supports a high-level non-procedural query language.

3. The storage and data structures are very simple; all data is represented in the form of records.

4. Access paths do not have to be predefined. A number of power operators are supported in relational model, e.g., select, project, join, etc. for data retrieval.

5. Because of the decline of hardware cost and the rise of manpower cost, a high-level nonprocedure manipulation language is necessary to minimize the user workload.

6. The relational model provide a simple and powerful interface to the data.

7. The relational model has fast response to ad hoc queries which are considered to be the high-percentage of queries.

8. The advance in associative storage devices offers the potential of greatly improving the efficiency and therefore the performance of a relational system.

Based on this choice, we propose a database communication system which incorporates distributed heterogeneous systems into a unified entity and shares the information resources in a distributed manner.

5.3.2 Architecture of Database Communication Systems

Although the heterogeneous database management systems are usually geographically distributed, the existing approach for communication between heterogeneous DBMSs builds a single control system which coorperates and communicates between different DBMSs by using the computer network. One asks why shouldn't the database control also spread through each coorperating DBMS? Hopefully doing so will provide a better use of data resources and improve the performance and reliability.

Our approach is to define a database communication system which serves as a front-end processor of local DBMSs as an interface to the computer network. It is a software

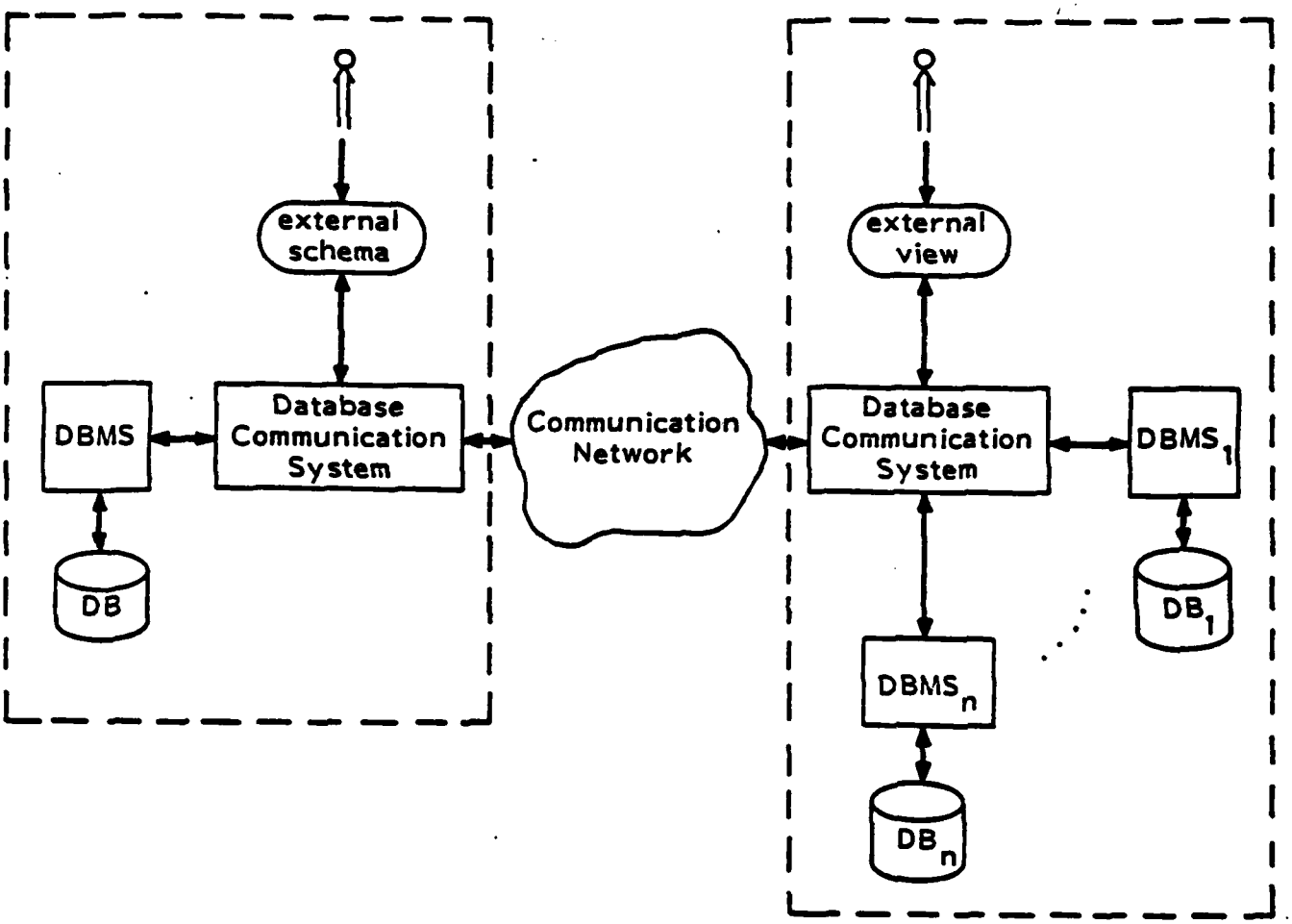


Fig. 5.2 General Architecture of Heterogeneous DBMSs.

system aimed to link geographically distributed heterogeneous DBMSs together and to act as a bridge for communication between local DBMSs (see Figure 5.2).

The basic underlying assumptions are:

1. It is possible to exchange information among the various systems and they are willing to maintain information.
2. Each DBMS is considered to be able to execute a given local transaction.
3. There exists a communication network which connects the various DBMSs.
4. The access to a local DBMS is not affected by the operation of the data communication system which should be transparent to the local user.

Functional Characteristics

The database communication system consists of three major units: (see Figure 5.3)

* Schema Unit
* Query Unit
* Control Unit

The functional characteristics of each component within a unit are described separately in order to maintain the modularity.

A. Schema Unit:

The schema unit maintains the local schema and integrity schema. It consists of three components. We will

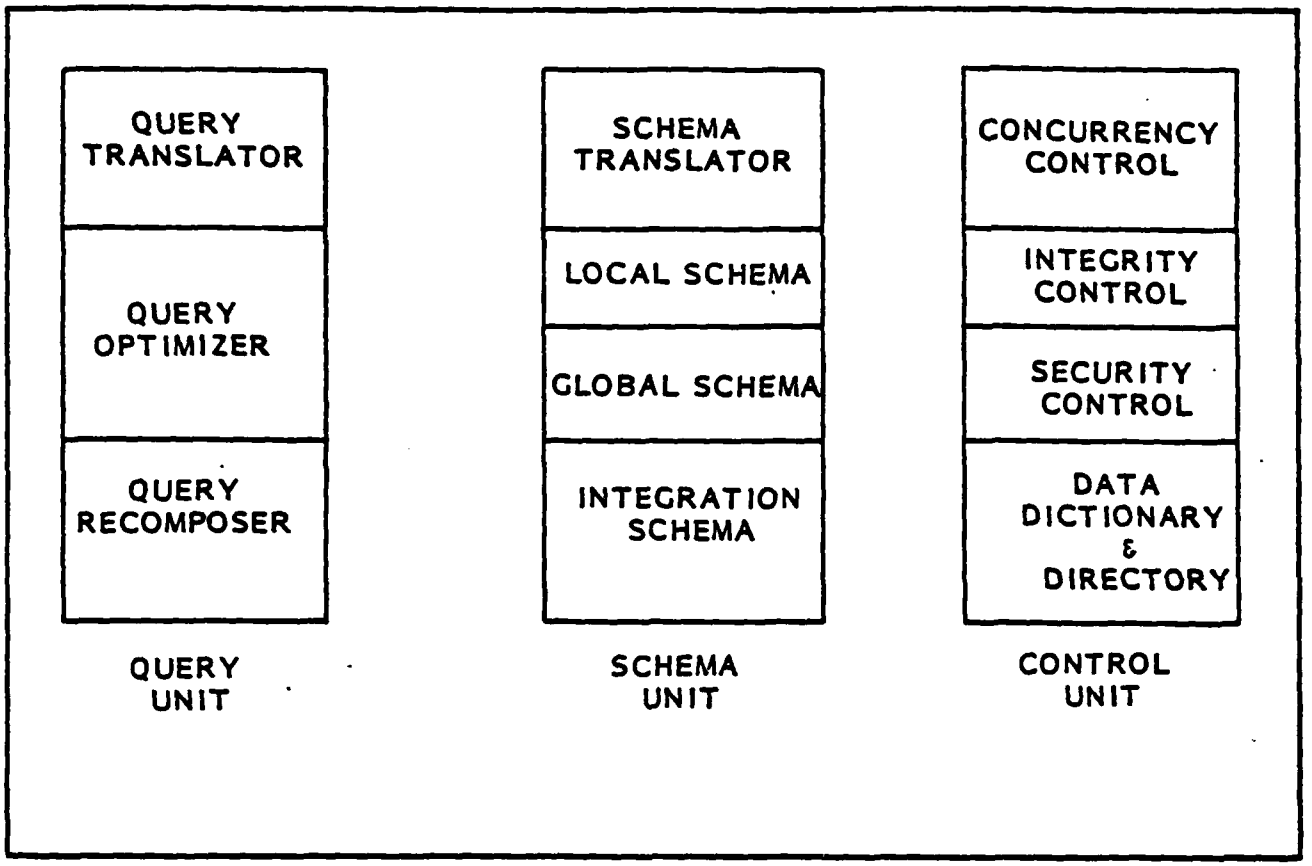


Figure 5.3 ARCHITECTURE OF DATABASE COMMUNICATION SYSTEM

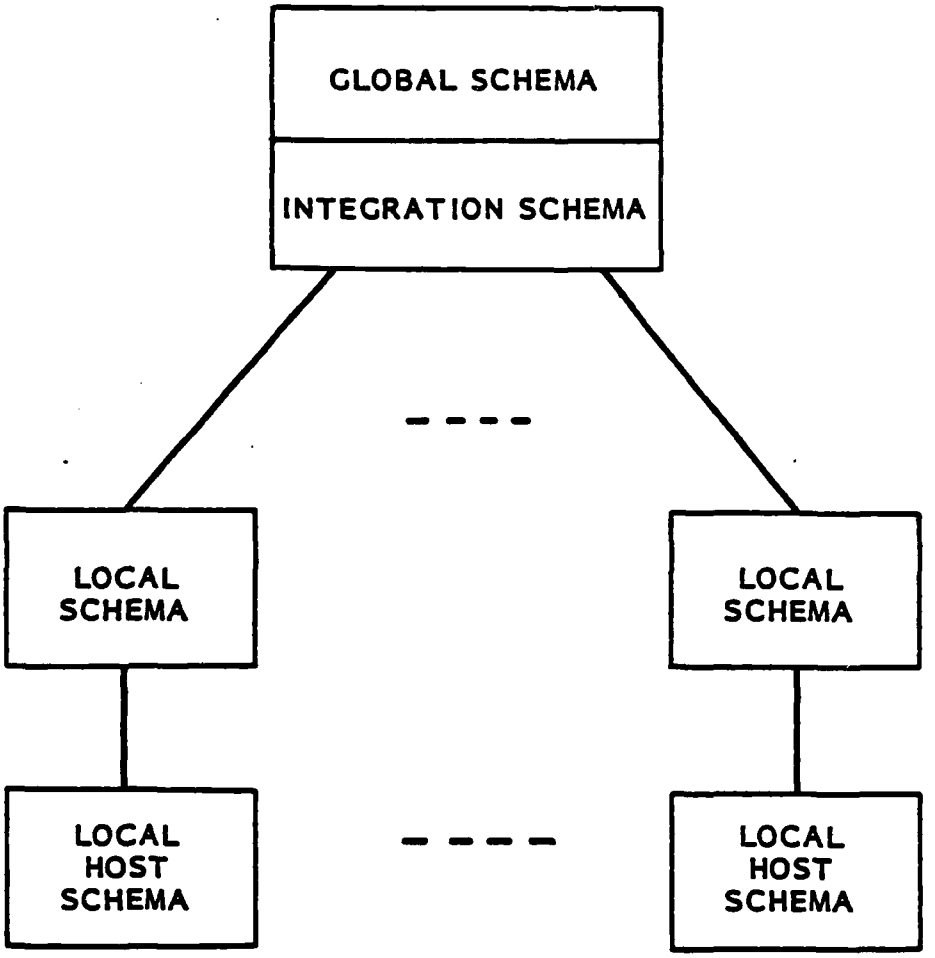


Figure 5.4 SCHEMA ARCHITECTURE

describe each component following by its important functions. (see Figure 5.4)

(A.1) Schema Translator

* Reads a schema description of the local DBMS and translates it into a schema description in a global data model and vice versa.
* This is done by a mapping of the data definition language and the structure of the data model.
* The schema translator may be different for different target DBMS.
* A schema unit can have several different kinds of schema translators.

(A.2) Local Schema and Global Schema

* Local schema is the schema translated by the schema translator from the local host schema.
* Global schema is the union of all local schema and integration schema of the database communication system.

(A.3) Integration Schema

* It consists of information about integrity constraints, data incompatibility, and data redundancy.
* It is set up at the time a DBMS joins to the heterogeneous network.
* The component can be viewed as a small

database.

B. Query Unit:

A query unit takes care of the query processing, optimization and access strategy. It consists of three components. (see Figure 5.5)

(B.1) Query Translator

* Translates a query in the global query language into a query accepted by the local DBMS.
* This is done by a mapping of the data manipulation language.
* The query is parsed and simplified.

(B.2) Query Optimizer

* The query is decomposed into local subqueries which reference only local schema and queries which reference only the integration schema.
* The distributed query algorithm must provide an execution strategy which minimizes both the amount of data moved from site to site and the number of messages sent between sites. In addition, the algorithm should take advantage of the computing power available at all of the sites involved in the processing of the query.
* The algorithm must also take care of the

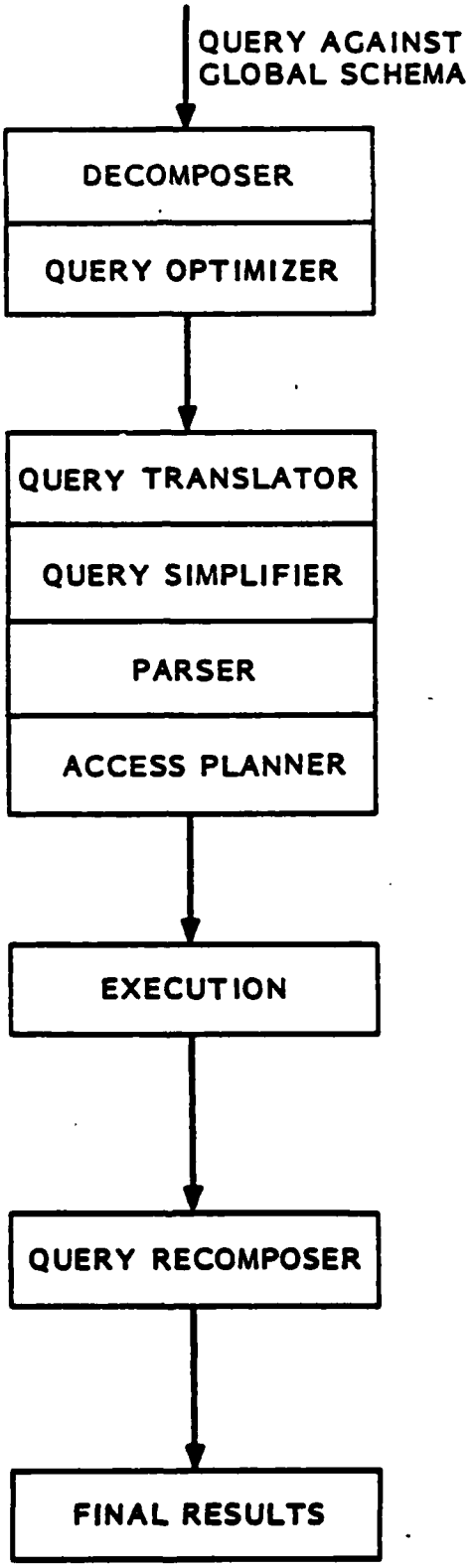


Figure 5.5 QUERY PROCESSING

query recomposer within the optimization strategy.

(B.3) Query Recomposer

* The access strategies are then executed. The results of the execution are represented in local host schema. The final answer must be described in terms of global schema.
* The result of local queries must be sent to the answer site so that the results can be put together and reformated as the answer expected by the query.

C. Control Unit:

(C.1) Concurrency Control

* The concurrency control algorithm must have a synchronization protocol to preserve consistency in a distributed environment.
* It processes distributed interleaved transaction by guaranteeing that all nodes in the system process the accepted update in the same reference.
* Deadlock detection or prevention mechanisms must be provided. When system failures occur, the other nodes must be able to continue to operate and the crashed nodes must be able to restore correct operation.

(C.2) Integrity Control

* There are two levels of consistency. Strong mutual consistency has all copies of data in the system updated at the same time. Weak mutual consistency allows various copies of the data to converge to the same update status over time, but, at any instant of time, some copies may be more up-to-date than the others.
* In a C operational system, we may want to adapt weak mutual consistency so as to use less processing time.

(C.3) Security Control

* All data, dictionary, programs and services must be protected from unauthorized access.
* All authorization information is kept locally and checked locally.
* A feedback encryption and decryption system must be provided to each node across the communication network.

(C.4) Data Dictionary/Directory Schema

* Provides user with a transparent view of the directory.
* Keeping information about where various data stored to efficiently provide the system query unit access data.

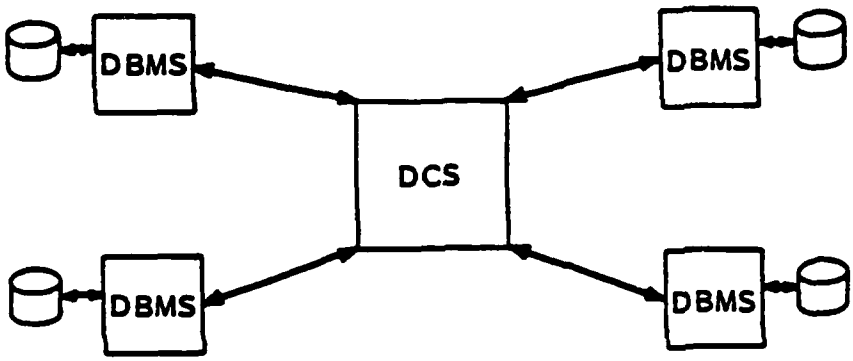
* Keeping one central master directory in the system with each local DBMS keeping a local subset of the control directory.
* These are the "bread and butter" software for a successful database administration function.

5.3.3 Heterogeneous DBMSs Network

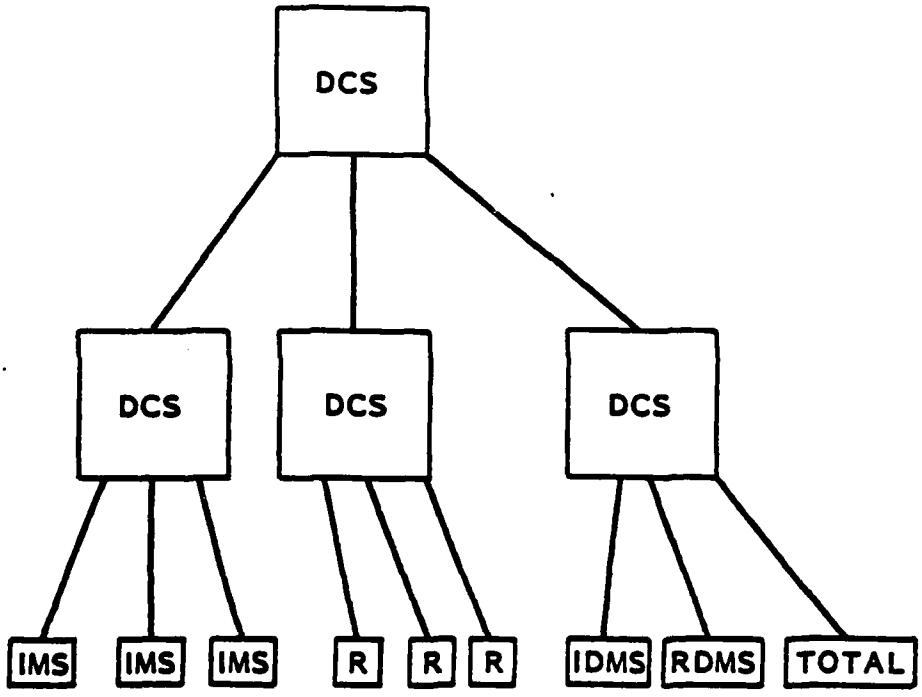
By using a database communication system (DCS), the heterogeneous DBMSs can be integrated in several interconnection configurations according to the desired criteria. For example, several versions of the same type of system could be grouped together under a local database communication system so that they can easily communicate without needing translation if a query just references the local DBMSs. The systems which store similar data can be grouped together at a first level so that they can be more efficient in retrieving data and exchanging information. Those systems which store confidential data can be put together so that the management and security control can be handled more effectively.

The heterogeneous DBMSs network using a database communication system as a bridge for interconnection may have one of the following configurations:

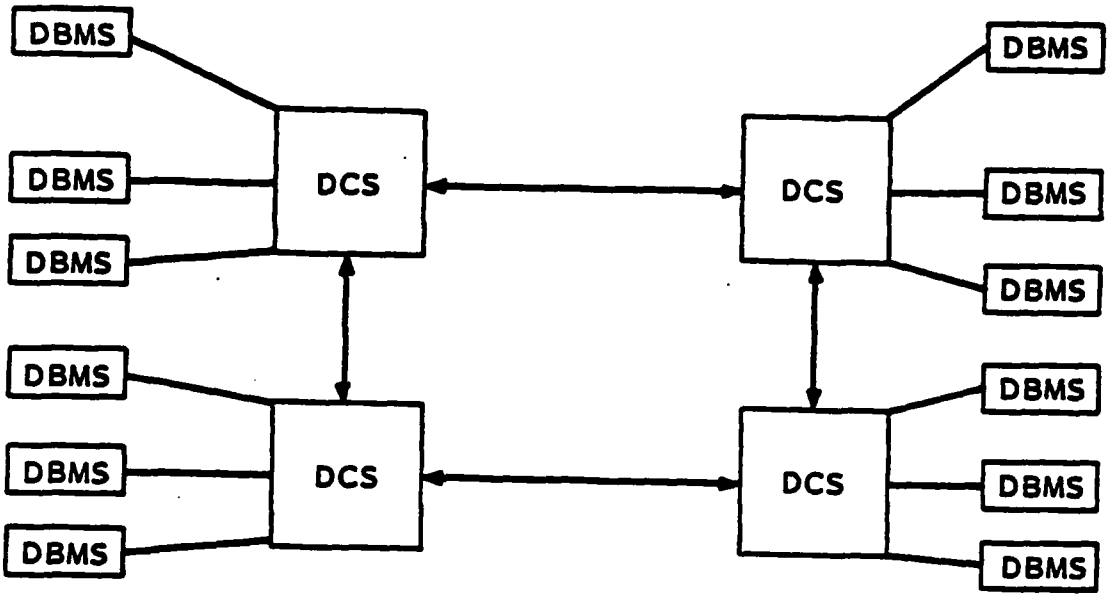
1. Star Architecture (Centralized System)



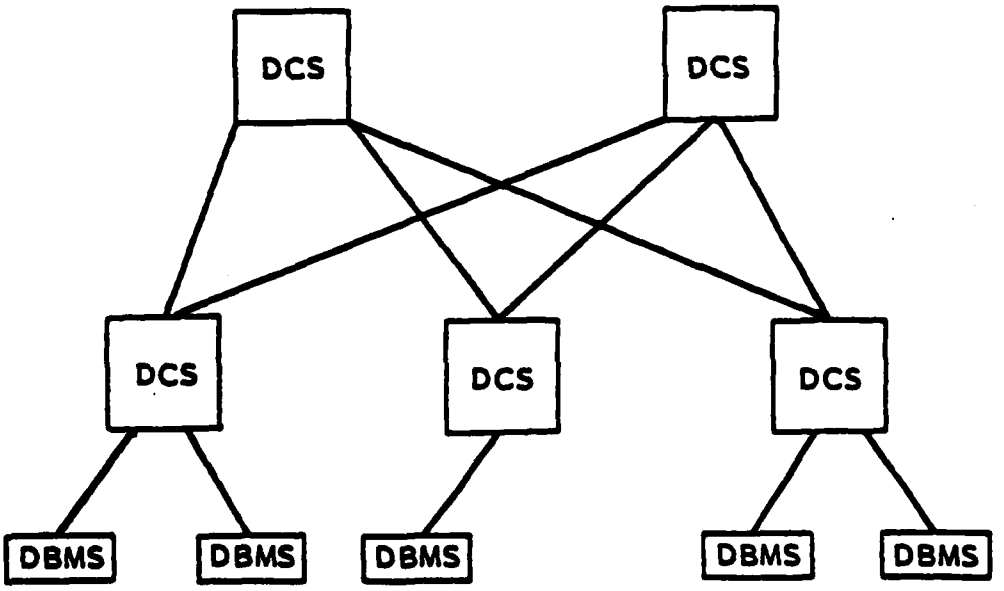
2. hierachical architecture



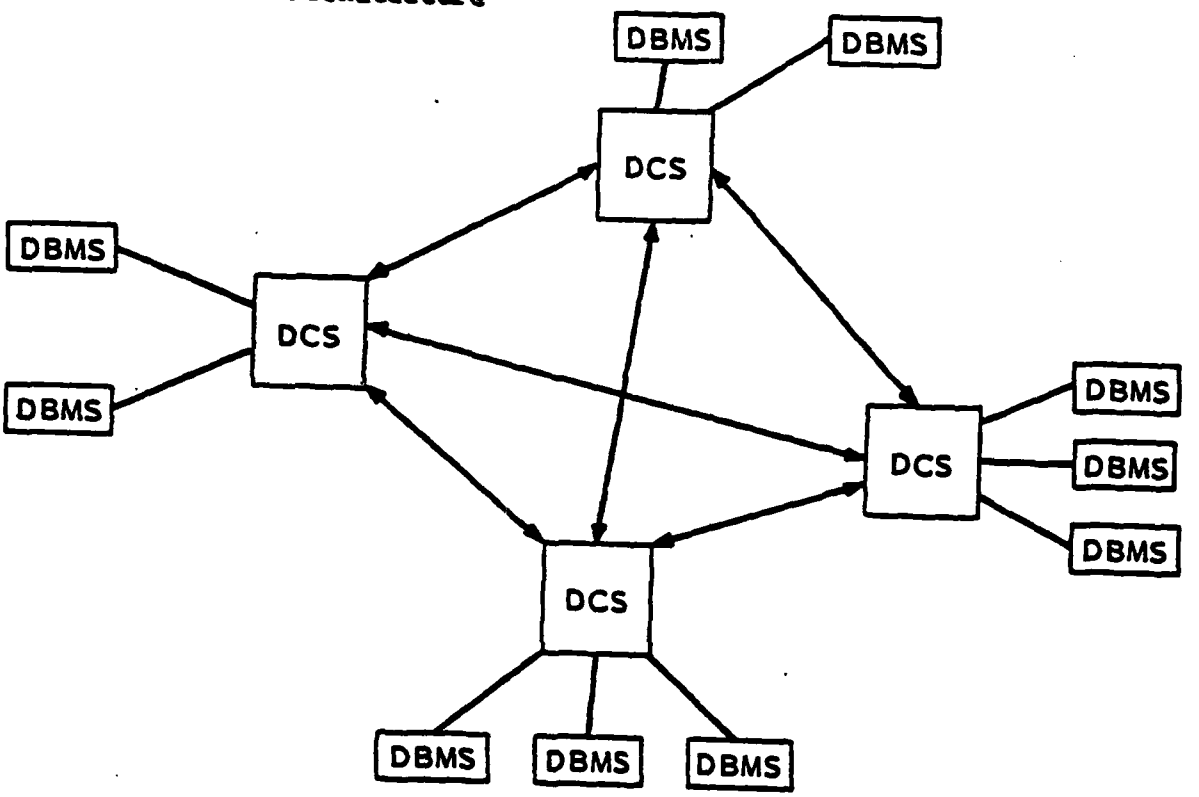
3. Ring architecture



4. Partition Architecture



5. General Architecture



Advances in hardware, software, and communication technology will make the distributed DBMSs possible. Several communication networks are commercially available to integrate the distributed DBMSs. For instance, ARPANET is one example of a point-to-point partially interconnected public data network. It is a packed-switched network which interconnects large-scale computers worldwide. Ethernet is one example of a branching broadcast communication system. It is a local computer network for carrying digital data packets among locally distributed computing stations.

It will also be possible to interconnect existing networks of multiple systems with new designs, using gateway technology. Combinations of loop or hierarchical and point-to-point interconnect technology will make it feasible to develop complex local/remote systems. The combination of low-cost satellite communication links and high-speed fiber-optics loop-based systems will also provide a basis for large, complex, hybrid interconnect structures to share information on distributed DBMSs.

5.4 Query Processing in a Heterogeneous Environment

Based on the architecture of the database communication system (DCS) we proposed in [HD81] , we adopt the relational model as the global conceptual model. It is necessary to provide a relational schema for each database. For those

databases in which the underlying data models are not relational models, schema translators will be required to do the translation jobs. A specific schema translator is needed for a specific data model to translate the underlying schema to a relational logical schema. The integration schema consists of information about integrity constraints, data incompatibility and data redundancy. This integration schema can be viewed as a small database. For the query which is against this local relational schema, it is also necessary to provide translation rules to translate the relational operations into data manipulation language statements of the underlying data model. The users will see the system as a distributed relational database system.

In our approach, each database system is presented to a user with a global relational schema. A query for data access or update is specified in terms of a relational calculus-like qualification over relations with a target list. Codd's data sublanguage ALPHA (DSL ALPHA) [CODD 72] is one of the calculus-based data sublanguages. It consists simply of the relational calculus in a syntatic form which more closely resembles that of a programming language. In practice, the syntax would have to be compatible with that of the host language, whatever that was. For our purpose, we shall use the syntax of ALPHA which is expressed in [CODD 70].

Given a relational Alpha query presented to the system, the optimizer in the query processing unit of the system will find an optimal strategy and transform the query into an optimal sequence of relational algebra operators. In order to execute the operations against the local DBMS, the query translator will have to translate a relation algebra operation against the local DBMS into a program of data manipulation language (DML) statements of the target system.

Join, Semijoin, Union, Intersection and Difference of two relations require two operands having the same set of attributes. They can be executed by using project and select operations to retrieve data and put it into relational form and then perform the desired binary operation. The details of query translation will be discussed in the next chapter.

5.5 Conclusions

We have described the architecture of a heterogeneous database communication system in this chapter. The database communication system is an approach to integrating heterogeneous database management systems. We believe that the integration of many independent, distributed information resources, should be helpful in information retrieval and decision making in solving real world problems. There are several problems which need further study in order to make the system successful: query optimization, distributed concurrency control, translation rules, and security

control, etc.

In the environment of business operations, time is a very important factor. The user should be able to easily form a query and have the system retrieve the data and recompose it for quick presentation to the user. Query optimization is the most important component and is the subject of this thesis. Concurrency control problems have been widely studied, mostly for centrally controlled systems. We need to study and develop algorithms which are suitable for a distributed environment. For the translation between global schema and local schema, and between a query in a global language and a query in a local language, we need to study the rules for translation for different data models and manipulation languages. This is the subject of the next chapter. Security control is one of the most important problems in DBMS systems. Because the data are often integrated together, the security control of classified information is essential. The mechanism for checking access rights and encryption of the information flowing throughout the network deserves further study. This solution of a database communication system makes the distributed system transparent to users from an operational point of view. It is hoped that such a database communication system will increase the efficient usage and management of information and data of heterogeneous database management systems.

In the next chapter, we will study query processing in heterogeneous database management systems. Schema translation and query translation will also be described.

Chapter 6
Schema and Query Translation

6.1 Introduction

Based on the architecture of the database communication system we proposed in the last chapter, we adopt here the relational model as the global conceptual model. It is necessary to provide a relational schema for each database management system (DBMS). For those databases in which the underlying data models are not relational models, schema translators will be necessary in order to provide the relational view to the user. For each specific data model, a specific schema translator is needed for a specific data model to translate the underlying schema to a relational logical schema. If the underlying data is a relational model, in some cases we still need to do some adjustment and provide a uniform relational schema. For example, in one database, the students' information relation will have attributes (course, faculty, studentID, grade), and in the other database, there will be a relation with attributes (course, professor, studentID, grade). The attribute "faculty" in one relation in fact has the same domain as the attribute "professor" in the other relation. We need to put this condition as an integrity constraint in the integrity schema component and provide a new uniform relation schema for this system. The other problem we also need to consider is the data incompatability. In the first relation, the

grade could be given by a number scale between 0 and 100 , and the second relation could have the grade in the literal scale "A" to "F." The data incompatability must be taken care of by the integrity schema. Schema translators are one of the topics that will be discussed in this chapter.

Data manipulation facilities will also be discussed in this chapter. In our heterogeneous database communication system, we use ALPHA, a non-procedural relational calculus like language, to express a query. Given a query, it will be analyzed by the query optimizer and be transformed to a sequence of relational algebra operators against those local relational schemata. Because the underlying data models are not necessary relational models, it is also necessary to provide translation rules to translate the relational operations into corresponding data manipulation language statements of the underlying data model. In this chapter, we address the problem of designing a schema translator and a query translator of a specific data model.

Relational, network, and hierarchical models are the three major data models that have been used in database systems. The relational model has been adopted here as the global conceptual model. We need to consider schema translators from relational model to relational model, from relational model to network model, and from relational model to hierarchical model. We also need to consider query translators from relational algebra operators to

corresponding data manipulation languages of the three data models. As we can see, the schema translator and the query translator for the case in which the underlying data model is relational are rather easy to develop. We leave this case here without any futher detailed specification. Also,since the present state of the art is such that the existing commercial systems are almost exclusively based on one of the other two models, we therefore will restrict our effort here to the design of translators for systems based on these two models.

Hierarchical systems provide for the same form of association between two records as does the network system. A hierarchy is simply a network that is a forest (a collection of trees) in which all links point in the direction from child to parent. If we have an hierarchical design we can thus clearly regard it as a particular network specification and will have no difficulty implementing it using a network-based software product. It is not difficult to see that similar remarks can be made about the schema translation and query translation. Because of this reason, we concentrate our effort in this chapter on schema translation from a network model to a relational model and query tranlation from relational algebra operators to network data manipulation languages.

We focus on those changes that must be made because of the difference in the level of procedurality of the

relational algebra operators and the data manipulation language of target data models. We do not consider the problem of modifying the program when the schema is altered due to database redesign and evalution. We also do not attempt to address the issues involved in the implementation of these translation rules, such as the syntax of the language used in a specific system.

This chapter is organized as follows. In the next section, schema translation rules are formulated for translating CODASYL/DBTG schema into relational schema. Based on this translation rule, we derive the query translation rules for mapping relational algebra operations into CODASYL data manipulation language statements in section 6.3.

6.2 Schema Translation

A great deal of attention has been focused on the network approach since the publication in April 1971 of the CODASYL DBTG final report [CODA 71]. The initial DBTG specifications have undergone subsequent development and refinement as reported in [CODA 73] & [CODA 78] by CODASYL groups. A number of commercially available systems have used one or more versions of the specifications as the implementation base. While those commercial implementations may show slight differences, their underlying concepts are based on the same CODASYL/DBTG data model.

The DBTG specifications propose three levels of data organization, associated data definition language, and the language for processing this data:

1. The schema data definition language, schema DDL.

2. The subschema data definition language, subschema DDl.

3. The data manipulation language, DML.

4. The device/media control language, DMCL.

The schema is the logical description of the data base. A schema description in the DBTG DDL includes four types of declarations.

1. The schema name description-- it is unique for each schema handled by the DBMS.

2. Record type declarations-- they define the data items for each record.

3. Set declarations-- they define the relationships between defined record types.

4. Area declarations-- they define the physical areas in which records will be stored.

There are two kinds of record types in the CODASYL/DBTG model: a description record type and a connection record type.

A description record type in the DBTG data model has a record ID, and one or several attributes describing properties of the record. It is very similar to a relation in the relation data model with the record ID as the key attribute. Therefore, a description record type can be translated to a relational schema directly.

A connection record type is introduced when n types of entity (represented by n description record types) are to be connected. N set types are also introduced. Each of the n "entity" record types is made the owner of one of the set types, and the connection record type is made the member of the set types; and each connection record occurrence is made a member of exactly one occurrence of each of the n types of set and thus represents the connection between the corresponding n entities. For this record type we define a relation schema R with attributes consisting of the keys of the owners of the n sets in which this record is a member and the data items of this record. The key of this relation is the set of keys of the owners of the n sets.

A set type is defined in the schema as having a certain type of record as its owner and some other type of record as its member. Each occurrence of a set type consists of precisely one occurrence of its owner together with zero or more occurrences of its member. For each set type, we do not correspondingly define a relational schema. Instead, we

create a table in each node to record all the sets which can be thought of as defining an access path relation. This table contains three attributes { set, owner, member }. It is used in the query translation process from relational operator to CODASYL DML statements to identify the access path. The table can be thought of as a new relation which is stored in the local database and only used for query translation. This table provides information of record access paths. It isn't joined to the global schema to be presented to the user. A singular set is a set with a system as an owner and a description record as a member. It can be thought of as a set having exactly one occurrence and having no owner record. By using the singular set construct, the set of description record occurrences is exactly like a sequential file. It provides the user an access path to access the description record. For the singular set, we also keep the set information on the table.

An area is a storage space of a DBTG database. For each type of record the schema specifies the area into which occurrences of the record are to be placed when they are entered into the database. This concept can be thought of as the vertical and horizontal partition of the database. For instance, consider a university application that creates student records. The database administrater may decide for a varity of reasons that instead of representing status as a data-item in the student record type, the classification is

to be made by storing student records in two distinct areas, graduate and undergraduate. This area type is correspondently mapped to the horizontal or vertical partition of relational database which may be necessary to create a new relational schema.

In this thesis, we shall assume that all occurrences of a given type of record are to go into a single area.

In conclusion, we summarize the translation rules in Figure 6.1, and give an example of the translation rules.

1. For a description record type R, we define a relation schema R', with each data item to be a attribute of R. The identifier of R is the key of R'.

2. For a connection record type, we define a relation schema S' with attributes consisting of the identifiers of the owners of the sets in which S is a member and the data items of S. The set of identifiers of the owners of the set is the key of S'.

3. For each set-type, we maintain a table which contain three attributes: set, owner and member.

Figure 6.1

EXAMPLE [DATE 77]

Assume the supplier-part-project database S is stored in a CODASYL version DBMS. The schema of this database is defined as follows.

```
SCHEMA NAME IS S-P-J-SCHEMA
AREA NAME IS S-AREA
AREA NAME IS P-AREA
AREA NAME IS J-AREA
AREA NAME IS SPJ-AREA

RECORD NAME IS S;
LOCATION MODE IS CALC HASH-SNO USING SNO IN S;
WITHIN S-AREA;
IDENTIFIER IS SNO IN S.
02 SNO    ; TYPE IS CHARACTER 4.
02 SNAME  ; TYPE IS CHARACTER 20.
02 STATUS  ; TYPE IS CHARACTER 3.
02 CITY   ; TYPE IS CHARACTER 15.

RECORD NAME IS P;
LOCATION MODE IS CALC HASH-PNO USING PNO IN P;
WITHIN P-AREA;
IDENTIFIER IS PNO IN P.
02 PNO    ; TYPE IS CHARACTER 4.
02 PNAME  ; TYPE IS CHARACTER 20.
02 COLOR   ; TYPE IS CHARACTER 6.
02 WEIGHT   ; TYPE IS CHARACTER 4.

RECORD NAME IS J;
LOCATION MODE IS CALC HASH-JNO USING JNO IN J;
WITHIN J-AREA;
IDENTIFIER IS JNO IN J.
```

```
      02 JNO    ; TYPE IS CHARACTER 4.
      02 JNAME  ; TYPE IS CHARACTER 20.
      02 CITY   ; TYPE IS CHARACTER 15.

RECORD NAME IS SPJ;
      LOCATION MODE IS SYSTEM-DEFAULT;
      WITHIN SPJ-AREA;
      IDENTIFIER IS SNO IS SPJ,
                    PNO IN SPJ,
                    JNO IN SPJ.
      02 SNO     ; TYPE IS CHARACTER 5.
      02 PNO     ; TYPE IS CHARACTER 6.
      02 JNO     ; TYPE IS CHARACTER 4.
      02 QTY     ; TYPE IS FIXED DECIMAL 5.

SET NAME IS S-SPJ;
     OWNER IS S;
     ORDER IS PERMANENT SORTED BY DEFINED KEYS.
     MEMBER IS SPJ
        INSERTION IS AUTOMATIC
        RETENTION IS MANDATORY;
        KEY IS ASCENDING PNO IN SPJ, JNO IN SPJ
            DUPLICATES ARE NOT ALLOWED
            NULL IS NOT ALLOWED;
        SET SELECTION IS THRU S-SPJ OWNER
            IDENTIFIED BY IDENTIFER SNO IS S
                          EQUAL TO SNO IN SPJ.
```

```
SET NAME IS S-SET;
    OWNER IS SYSTEM;
    ORDER IS PERMANENT SORTED BY DEFINED KEYS.
    MEMBER IS S
       INSERTION IS AUTOMATIC
       RETENTION IS FIXED;
       KEY IS ASCENDING SNO IN S
           DUPLICATEs ARE NOT ALLOWED
           NULL IS NOT ALLOWED;
       SET SELECTION IS THRU S-SET SYSTEM.

 SET NAME IS P-SPJ;
     OWNER IS P;
     ORDER IS PERMANENT SORTED BY DEFINED KEYS.
     MEMBER IS SPJ
        INSERTION IS AUTOMATIC
        RETENTION IS MANDATORY;
        KEY IS ASCENDING JNO IN SPJ, SNO IN SPJ
            DUPLICATES ARE NOT ALLOWED
            NULL IS NOT ALLOWED;
        SET SELECTION IS THRU P-SPJ OWNER
            IDENTIFIED BY IDENTIFER PNO IN P
                          EQUAL TO PNO IN SPJ.

  SET NAME IS P-SET;
      OWNER IS SYSTEM;
      ORDER IS PERMANENT SORTED BY DEFINED KEYS.
```

```
    MEMBER IS P
       INSERTION IS AUTOMATIC
       RETENTION IS FIXED;
       KEY IS ASCENDING PNO IN P
           DUPLICATES ARE NOT ALLOWED
           NULL IS NOT ALLOWED;
       SET SELECTION IS THRU P-SET SYSTEM.

SET NAME IS J-SPJ;
    OWNER IS J;
    ORDER IS PERMANENT SORTED BY DEFINED KEYS.
    MEMBER IS SPJ
       INSERTION IS AUTOMATIC
       RETENTION IS MANDATORY;
       KEY IS ASCENDING SNO IN SPJ, PNO IN SPJ
           DUPLICATES ARE NOT ALLOWED
           NULL IS NOT ALLOWED;
       SET SELECTION IS THRU J-SPJ OWNER
           IDENTIFIED BY IDENTIFIER JNO IN J
                         EQUAL TO JNO IN SPJ.

SET NAME IS J-SET;
    OWNER IS SYSTEM;
    ORDER IS PERMANENT SORTED BY DEFINED KEYS.
    MEMBER IS J
       INSERTION IS AUTOMATIC
       RETENTION IS FIXED;
```

```
KEY IS ASCENDING JNO IN J
    DUPLICATES ARE NOT ALLOWED
    NULL IS NOT ALLOWED;
SET SELECTION IS THRU J-SET SYSTEM.
```

After applying the rules of schema translation, we define the following relational schema

S=(SNO, SNAME, STATUS, CITY)

P=(PNO, PNAME, COLOR, WEIGHT)

J=(JNO, JNAME, CITY)

SPJ=(SNO, PNO, JNO, QTT).

We create an access path relation as follows:

set	owner	member
S-SPJ	S	SPJ
J-SPJ	J	SPJ
P-SPJ	P	SPJ
S	SYSTEM	S
P	SYSTEM	P
J	SYSTEM	J

6.3 Query Translation

In our database communication system, each database system is presented to a user with a global relational schema. A query for data access or update is specified in terms of a relational calculus-like qualification over relations with a target list. Codd's data sublanguage ALPHA (DSL ALPHA) [CODD 72] is one of the calculus-based data sublanguages. It consists simply of the relational calculus in a syntactic form which more closely resembles that of a programming language. In practice, the syntax would have to

be compatible with that of the host language, whatever that was.

For our purpose, we shall use the syntax of ALPHA which is expressed in [CODD 70]. We restrict ourselves to consideration of its major features only. Assume a query Q express in ALPHA issued at site S. An example of a query expressed in ALPHA is:

EXAMPLE:

Let the query be "Get SNO values for suppliers who supply a LONDON or PARIS project with a red part."

The corresponding ALPHA statement of this query will be

```
RANGE P PX
RANGE J JX
GET W (SPJ.SNO) : ∃ PX ∃ JX (PX.COLOR='RED'
                  ∧(JX.CITY='LONDON'∨JX.CITY='PARIS')
                  ∧ SPJ.PNO=PX.PNO ∧ SPJ.JNO=JX.JNO)
```

The list of attributes within parenthesis of W is the target list which specifies the attributes to retrieve. The predicate calculus following ":" is the qualification

The query optimizer in the query processing unit of the system will transform the query into a sequence of relational algebra operations. This sequence of relational algebra operations is the optimal query processing strategy. Some of the DBMSs may be implemented on the CODASYL model.

In order to execute the operations against the local DBMS, we have to translate a relation algebra operation against this DBMS to a program of CODASYL DML statements. The query translator of this section plays the role of this process.

CODASYL Data Manipulation Language

We briefly introduce the way queries are specified in CODASYL DML. In the CODASYL model, the major DML statements are the following:

FIND--------locates an existing record occurrence and establishes it as the current of run-unit.

GET---------retrieves the current of run-unit.

MODIFY------updates the current of run-unit.

CONNECT-----inserts the current of run-unit into one or more set occurrences.

DISCONNECT--removes the current of run-unit from one or more set ocurrences.

ERASE-------deletes the current of run-unit.

STORE-------creates a new record occurence and establishes it as the current of run-unit.

A query in the CODASYL model is a sequence of DML statements which are embedded within a program as a syntactic extension of the host language. The function of the FIND statement is to locate a record occurrence in the database and to make it the current of the run-unit (the most recently accessed record occurrence), also the current of the appropriate area (logical partitioning of the

database), the current of the appropriate record type, and the current of all sets in which it participates. It is logically required before each of the other statements, except STORE. Note that the FIND statement itself does not retrieve any data. For each program operating under its control, the DBMS maintains a table of "currency status indicators" in the user working area (UWA). These indicators are actually database-key values.

The current of the run-unit, whatever its type, can be brought into the UWA only by executing GET. The modify statement replaces (portions of)the current of the run-unit with values taken from the UWA. The STORE statement will store newly constructed data items in the UWA into the database.

In the distributed DBMS environment, we assume each system has a SEND command which can send a file or part of the UWA from one system to another system.

To update a database, or to create a new record, it is necessary to retrieve the data occurrence to be updated or to create a new data occurrence in the UWA. After updating the data occurrence in the UWA, a MODIFY or STORE statement is then applied. Therefore, we need to consider only retrieve operations.

We first study the translation procedures for two unary algebra operators: project and select.

We assume $R=\{A_1, A_2, \ldots, A_n\}$ and $X=\{A_1, \ldots, A_K\}$. PROJECT, denoted by $\Pi_X(R)$, is to retrieve the attributes in X of each record in R. The algorithm $PROJECT_N(R,X)$, as shown below, is to translate $\Pi_X(R)$ into a DML program.

```
PROJECT_N(R,X):
   IF ( Π_Set(t)='R-set' and Π_Owner(t)='system')
       /* description record */
   THEN      NXT:  FIND next R within R-set;
                   IF end of set GOTO quit;
                   GET A1 IN R, ...., AK IN R;
                   GOTO NXT.
   ELSE IF ( Π_Set(t)='S-R' and Π_Owner(t)='S')
          /* connection record */
        THEN     NXT: FIND next R within S-R;
                      IF end of set GOTO quit;
                      GET A1 in R,..., AK in R.
                      GOTO NXT.
```

Let $Y=\{A_1, A_2, \ldots, A_\ell\} \subseteq R$. SELECT, denoted by $\sigma_{\{X A_i='c_i'\}}(R)$, is to select tuples in R such that $R.A_1='c_1', \ldots, R.A_K='c_K'$. The algorithm below is the translation algorithm.

```
SELECT(R,Y,c1,c2,...,cK):
   IF (R is a connection record)
   THEN if (some Ai is and attribute of S) & (S-R is a set)
      THEN BEGIN
```

```
        MOVE 'c1' to A1 in S;
        FIND any S;
        IF S-R empty GOTO quit;
        MOVE 'c2' to A2 in R;
              :
              :
        MOVE 'ck' to Ak in R;
        FIND any R within S-R;
        IF end of set GOTO quit;
        GET R;
        END;
ELSE BEGIN
     /* there exist some S s.t. S-R is a set */
    NXT: FIND next S within S-set;
         IF end of set GOTO quit;
         GET A  in S;
         MOVE  'c1' to A1 in R;
               :
               :
         MOVE  'ck' to Ak in R;
         FIND any R within S-R;
         GET A1 in R, ... , Al in R;
         IF end of set GOTO NXT;
     END;
```

Next, we look at binary relation algebra operations. Several binary relation algebra operations (e.g., division, join, semijoin, etc.) which reference two relations may be stored at the same system. In this case, data must be sent from one system to another. For a CODASYL DBTG model DBMS, there are three strategies for sending data through the network: record-at-a-time transmission, set-at-a-time transmission and all-records-at-a-time transmission. In a CODASYL DBTG model DBMS, record-at-a-time and set-at-a-time access operators are supported which enable one node to send data to the other node after performing such an operation. The data transmission of these two types of operations requires considerable communication overhead. In a relational DBMS, all records required by another system are usually transmitted at a time.

We assume in this thesis that the transmission mode of any system will be all-records-at-a-time. We also assume that the data are in relational table-like form after being retrieved from local DBMSs and temporarily stored in the UWA of each local DBMS. We assume each DCS has the ability to execute relational algebra operations. This assumption will enable easier data transmission when it is required. It also makes the translation procedures of binary relational algebra operators distinct between the two cases in which the operands of the operator are at the same system and where they are not.

When the two operands of the operator are in different systems, we always retrieve the first operands and store them in the user working area as a relation and then send it to the other system to produce the final result. We will first consider the case when two operands are in different sites.

Two of the most useful distributed binary relational algebra operators for distributed query processing and the execution sequence are join and semijoin, defined below:

Let R_1 and R_2 be two relation schemas at different systems S_1 & S_2 respectively and $X=R_1 \cap R_2$ be the set of attributes in R_1 and R_2 . Without lost of generality, we assume $X=\{A_1, A_2, \ldots, A_K\}$. Let T_1 and T_2 be the temporary relations.

1. JOIN

DEFINITION:

The distributed join of R_1 and R_2 over X is a distributed query operator which executes the following sequence of operations:

1. retrieve R_2 from S_2 as T_2;
2. send T_2 from S_2 to S_1;
3. $R \underset{X}{|X|} T_2$ at site S_1.

The query translation of this operator is as follow:

IF R_1 and R_2 are both relational DBMSs

THEN execute it as relational operator;

ELSE IF R_1 is relational and R_2 is CODASYL

THEN

$P_N(R_2,R_2)$ at S_2 as T ;

send T from S_2 to S_1;

execute $R_1 |X|_X R_2$ as relational operator;

ELSE IF R_1 is CODASYL

THEN

retrieve R_2 from S_2 as a elation T;

send T from S_2 to S_1;

For all $t \in T$

$SELECT_N(R_1,R_1\text{-}X,t_1,t_2,\ldots,t_K)$;

This operation will create a new relation T' with relation schema R_1 U R_2 at the working area of system S_1.

2. SEMIJOIN

DEFINITION:

The distributed semijoin of R_1 and R_2 over X, $R_1 <|X| R_2$, execute the following sequence of operations:

1. project R_1 over X at S_1 and form a result T_1;
2. send T_1 to S_2;
3. join T_1 with R_2 at S_2 and form a result T_2, i.e. select R_2 based on T_1 at S_2;
4. send T_2 to S_1;
5. join R_2 to R_1 at S_1 to form a result T_3 at S_1, i.e. select R_1 based on X columns of R_2.

This operation can be executed as follows:

IF R_1 and R_2 are both relational systems
THEN execute the operator as relational operator;
ELSE IF R_1 is relational & R_2 is CODASYL
THEN
project R_1 on X at S_1 as T_1;
send T_1 to S_2;
for each $(t_1, t_2, \ldots, t_K) \in T_1$
$SELECT_N(R_2, R_2, t_1, \ldots, t_K)$
and form result as T_2;
send T_2 to S_1;
join R_2 to R_1 at S_1 as relational
operator;
ELSE IF R_1 is CODASYL & R_2 is relational
THEN
$PROJECT_N(R_1, X)$ as T_1;
send T_1 to S_2;
join T_1 with R_2 as relational
operator to get T_2;
send T_2 to S_1;
for $t=(t_1, \ldots, t_K, \ldots, t_m) \in T$
$SELECT_N(R_1, R_1, t_K)$
and form result as T_2;
ELSE IF both R_1 & R_2 are CODASYL
THEN
$PROJECT_N(R_1, X)$ as T_1;

send T_1 to S_2;
for each $t \in T_1$
  SELECT (R_2,R_2,t)
  and form result as T_2;
send T_2 to S_1;
for all $t=(t_1,\ldots,t_K,t_{K+1},\ldots,t_M)$
  T_2, SELECT (R_1,R_1,(t_1,...,t_K))
  form a result T_3;

The Union, Intersection and Difference of two relations require two operands having the same set of attributes. If both two operands are stored in relational systems, then we just perform the relational operation. If one of the two operands R are stored in CODASYL systems, we first use $PROJECT_N(R,R)$ to retrieve R to form a relation and then perform the relational operation afterward. If both operands are stored in CODASYL systems, then we must consider two cases. If the results of the operation are temporarily stored in the UWA for use by later operations, then we use both $PROJECT_N(R_1,R_1)$ and $PROJECT_N(R_2,R_2)$ to retrieve both two operands as two relations and send one relation from one site to the other to perform the operation; or we can use PROJECT to retrieve one operand as a relation and send it to the other site and then perform selections.

The second case is that in which the results of the operation are to be stored in one site. In this case, we use PROJECT to retrieve one operand as a relation and send it to the other site and perform corresponding CODASYL DML statements.

When the two operands of a binary operation are at the same site and are stored in a CODASYL system, then we can perform the operation in the same sequence we discussed above. However, we can do it in a more efficient way because it is not necessary to fisrt retrieve one operand as a relation in the execution of this operation. We can use solely CODASYL DML statements to perform this operation and do it by record-at-a-time or set-at-a-time. In some cases, we even can do much better by combining a sequence of relational opeations which reference data stored at the same CODASYL system and then translating them to a sequence of CODASYL DML statements.

Let us look at two examples: We assume S, P, J, SPJ are as in example 1 and they are stored at the same system.

EXAMPLE 2 Perform the operation J $\underset{JNO}{|X|}$ SPJ .

We can translate this operation into a CODASYL DML statements as follow:

```
NXT:  FIND next J within J-SET;
      IF end of set GOTO quit;
      get SPJ and form a new relation;
```

```
            GOTO NXT.
```

EXAMPLE 3

Let the subquery expressed in ALPHA referenced to this CODASYL DBMS be

```
PROJECT (JOIN (SELECT SPJ WHERE JNO='J1')
            AND
              (SELECT P WHERE COLOR='RED')
            OVER PNO) OVER SNO SNAME STATUS
```

The interpretation of this subquery is to get SNO, SNAME, STATUS values for suppliers who supply project J1 with a red part.

The corresponding CODASYL DML statement for this subquery could be

```
     MOVE 'J1' TO JNO IN J;
     FIND ANY J;
     IF J-SPJ EMPTY GOTO QUIT;
     MOVE BLANK TO TEMP-SNO;
NXT. FIND NEXT SPJ WITHIN J-SPJ;
     IF end of set GOTO QUIT;
     GET SPJ
     IF SNO IN SPJ = TEMP-SNO GOTO NXT;
     MOVE SNO IN SPJ TO TEMP-SNO;
     FIND OWNER WITHIN P-SPJ;
     GET P;
```

```
IF COLOR IN P ='RED'
   THEN
          MOVE TEMP-SNO TO SNO IN S;
          FIND ANY S;
          GET SNAME IN S, STATUS IN S;
          ADD TEMP-SNO, SNAME, STATUS
          VALUES TO RESULT LIST;
   ELSE   MOVE BLANK TO TEMP-SNO;
          GOTO NXT.
```

From these two examples, we learn that some optimization can be done on the translation of subquery referencing to a CODASYL DBMS into a sequence of CODASYL DML statements. This is one of the optimization problems of local processing. We do not plan to discuss it further within this thesis.

6.4 Conclusions

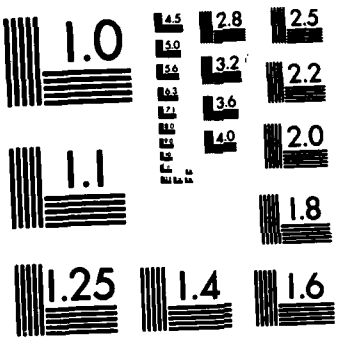
In this chapter, we have presented detailed schema translation rules for translating a CODASYL data model to a relational data model, and algorithms for translating relational ALPHA query into CODASYL DML statements whose associated databases schemata have themselves been translated. The translation algorithms are developed for each relational atomic operation. The translation procedures are based on the relational operations sequence provided by the query optimizer with the objective of minimizing

AD-A124 921 QUERY OPTIMIZATION IN DISTRIBUTED DATABASES(U) 3/3
MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR
INFORMATION AND DECISION SYSTEMS K HUANG OCT 82
UNCLASSIFIED LIDS-TH-1247 N00014-77-C-0532 F/G 12/1 NL

END

FILMED

DTIC



communication complexity and are done one relational algebra operation at a time. The translation procedures also use the information of the access path relation.

Some optimization for the local query processing can be done by translating a subquery which is a continuing subsequence of relational operations which reference data at the same DBMS into a sequence of CODASYL DML statements rather than translating one operation at a time. Because our major concern is the communication cost, this local processing optimization will not be considered in this thesis.

Chapter 7

Summary and Future Research

This thesis addresses the problem of query processing in distributed database management systems. In distributed homogenenous relational databases, we develop a mathematical model and algorithms for solving a query by a sequence of joins and semijoins using mixed strategy. We extend this model to distributed heterogeneous databases and only consider problems of schema translation and query translation which are parts of the query processing problem. Many other problems, e.g., consistency, redundancy, concurrent control, security, etc., remain to be done. We will here briefly summarize our research effort and suggest the potential direction of future research for each chapter and then make a few concluding remarks.

7.1 Summary and Future Research

In chapter 1, we discussed the need for research in query processing in the integration of distributed database management systems. That chapter also contains a goal and a road map for the entire thesis.

In chapter 2, we first reviewed some formal definitions of relational terminologies and previous studies of the query processing problem in distributed relational databases. Under the assumption that network traffic constitutes the main critical factor, we have developed a

mathematical model to compute the minimum communication cost of a join-semijoin program for processing a given equi-join query . For each equi-join query, we define a query processing graph for a given query from which the set of join-semijoin programs that solve the query can be characterized. A rule for estimating the size of the derived relation is proposed. The parameters for estimating the size of the derived relation form a consistent parameter system. With the assumption that the communication cost dominates and that the cost functions are linear in the sizes of the data to be transmitted, the distributed query processing problem is formulated as a dynamic network problem. A slight change of the model made by associating each node with a local processing cost will extend the model to the case where local computer processing time and transmission delay through the network are comparable. This could happen for distributed databases in a local area network environment, for example.

One future research direction is to extend this model to cover a larger class of queries,e.g., the class of ineqality-join queries, queries with existential quatifier, etc., by providing a model of measuring the reducibilities and estimating the size of the derived relations. Another potential research direction is the optimization of the local processing cost. In the case where both local processing costs and communication costs are important, the optimization of data retrieval when local computer

processing is used becomes essential. The access paths to the stored data, the implementation algorithms of relational algebra operations, e.g., join, project, selection, etc., and the order of relational algebra operations are also critical factors in query optimization. It would be very nice to provide a model to consider all those factors as well as communication costs.

In chapter 3, we studied the computational complexity of the distributed query processng problem. For a simpler case where all semijoin reducibilities are zero and join reducibilities do not affected by join operation, we have shown that under three different objective functions the problems of finding a routing strategy of required data to the site where a query is initiated are NP-complete. This gives us an indication that the distributed query processing problem is a hard problem. In fact, we can see from the nature of the dynamic network problem how to identify the reason for the difficulty of this problem. Future research direction will be the complexity of the query processing problem in the model of chapter 2.

One of the future research directions is to identify a set of conditions so that under those assumptions, the problem will be solvable using a polynomial time algorithm. Another future research direction is to find the computational complexity of an approximation problem for the distributed query processing problem. We conjecture that the

approximation problems are also very hard problems.

In chapter 4, we analyzed the difficult nature of the query processing problem and provided an analytical basis for heuristic algorithms. We first considered the simpler case of the problem where all possible semijoins are performed first, i.e. all semijoin reducibilities become zero. We provided several heuristic algorithms for this problem. Each of the algorithms has two stages. The first stage is to find a feasible processing strategy. The second stage is the improvement stage where interchange procedures are used. We then extend those heuristic algorithms to the general query processing problem by including semijoin operations into the sequence of join operations.

A future research direction is the study of the analytic behavior of those heuristic algorithms. Although some analytic results of worst case and average case analysis are difficult to obtain, some computational experiments may be conducted to get a feeling for the average performance of the algorithms. Developing more efficient and general solution procedures for general query processing problems is also a future research area.

In chapter 5, we developed a method for query processing in a distributed heterogeneous database management systems environment. A heterogeneous database communication system is proposed to integrate heterogeneous database management systems to combine and share

information. The architecture of a heterogeneous database communication system is described and several components are identified. The use of a database communication system for heterogeneous DBMSs makes the overall system transparent to users from an operational point of view. In this chapter, we are concerned only with query processing in a heterogeneous environment. Other problems of a database communication system such as concurrency control, updating, redundancy, security, etc., are subjects for future research.

In chapter 6, we presented detailed schema translation rules for translating a schema of a CODASYL data model to a relational schema. We also presented algorithms for translating relational algebra operations into CODASYL DML statements whose associated database schemata have themselves been translated. Translation algorithms are developed for each relational atomic operation. Relational, network, and hierarchical models are the major three data models that have been used in commercial database management systems. The schema translator and query translator for the case in which the underlying data model is relational are rather easy to develop. For the case where the underlying data model is hierarchical, the translation procedures are similar to the case of the network model.

Some optimization for the local query processing can be made by translating a subquery, which is a continuing subsequence of relational operations which reference data at

the same DBMS into a sequence of CODASYL DML statements rather than translating one operation at a time. In the case where the local processing cost is comparable to the communication cost, this optimization is essential and must be taken into account. This is a rich subject for future research.

7.2 Conclusions

The main objective of this thesis has been to study query processing in a distributed database environment. Different data retrieval strategies generally lead to substantially different system performances in terms of response time, computer utilization and network traffic. In practice, communication cost constitutes a major factor. Past experience showed that deciding a solution strategy for processing a given query is a very complicated problem. The mathematical formulation of the distributed query processing of this thesis provides a formal model of this problem. and we showed theoretically that this problem is indeed a very difficult problem. The heuristic algorithms proposed in this thesis, based on the analysis of the problem, should help in finding an optimization strategy for a distributed query processing environment.

The database communication system approach in chapter 5 provides a method of integrating heterogeneous database management systems. It leaves several problems for future

research activities. Detailed study and specification of each component of the system are desired. Future research efforts should also be oriented toward prototype system implementation.

Finally, it is hoped that more applications of the works in this thesis will occur in the future.

REFERENCES

[AB 74]
Abrial, J. R. Data semantics. IFIP-TC2 Working Conf., Jan 1976.

[AD 77]
Abida, M. and Delobel, C. The cooperation problem between different data base management systems. IFIP-TC2 Working Conf. Jan 1977.

[AD 78]
Abida, M. and Portal, D. A cooperation system for heterogeneous data base management systems. Inform. Systems 3(3):209-215, 1978.

[BC 81]
Bernstein, P. A. and Chiu, D. M. Using semi-joins to solve relational queries. J. ACM 28(1):25-40, Jan 1981.

[BFMMUY 81]
Beeri, C. etc. Properties of acyclic database schemas. 13th. ACM STOC 335-362 May 1981.

[BG 81]
Bernstein, P. A. and Goodman, N. The power of natural semijoins. SIAM J. of Comput. 10(4). Nov 1981.

[BG 80]
Bernstein, P. A. and Goodman, N. The power of inequality semijoins. TR-12-80, Aiken Computation Lab. Harvard Uuiv. Aug 1980.

[CH 76]
Chen, P. The entity relationship model: toward a unified view of data. ACM TODS 1(1) 1976.

[CH 80]
Chiu, D. M. and Ho, Y. C. A methodology for interpreting tree queries into optimal semi-join expressions, Proc. SIGMOD Conf Jun 1980.

[CHIU 79]
Chiu, D. M. Optimal query interpretation for distributed databases. Ph. D. thesis, Harvard Univ. 1979.

[CODA 71]
Data Base Task Goup of CODASYL programming language committee, Report, ACM Apr 1971.

[CODA 73]
CODASYL data description language committee, DDL Journal of Development 1973.

[CODA 78]
CODASYL data description language committee, DDL Journal of Development 1978.

[CODD 70]
Codd, E. F. A relational model for large shared data bases. Comm. ACM 13(6):377-387, Jun 1970.

[CODD 72]
Codd, E. F. Relational completeness of data base sublanguages. In Data Base Systems, Courant Computer Science Symposia Series, Vol. 6: 65-90, 1972.

[CP 80]
Cardenas, A. and Pirahesh, M. H. Data base communication in a heterogeneous data base management system network. Inform. Systems 5(1):55-79, 1980.

[DAT 77]
Date, C. J. An Introduction to Database Systems, 2ed, Addison Wesley, Reading Mass. 1977.

[DAY 79]
Dayal, U. Schema mapping problems in database systems, TR-11-79, Center of Research in Computing Technology, Harvard Univ. Aug. 1979.

[DL 80]
Delobel, C. and Litwin, W. Distributed Data Bases, Proc. of the international symposium on

distributed data bases French, March 1980.

[DP 80]
Draffan,I.W. and Poole, F, Distributed Data Bases, editor, Cambridge University Press 1980.

[ESW 78]
Epstein, R. etc. Distributed query processing in a relational data base system, Proc. SIGMOD Conf. 169-180 Jun 1978.

[GBWRR 81]
Goodman, N., Bernstein, P. A., Wong, E., Reeve, C. L., Rothnie, J. B. Query Processing in SDD-1. ACM TODS Dec. 1981.

[GRA 79]
Graham, M. H. On the universal relation. Tech. report, Univ. of Toronto, sep 1980.

[GS 81]
Goodman, N. and Shmueli, O. Syntactic characterizations of database schemas TR-09-81, Aiken Comp. Lab. Harvard Univ. Jun 1981.P

[HUL 81]
Hull, R. Acyclic join dependencies and database projections, In proc. XP2. June 1981.

[HY 79]
Hevner, A. R. and Yao, S. B. Query processing in distributed database systems. IEEE Trans. on Software Engineering, SE-5(3):177-187, May 1979.

[KER 76]
Kerschberg, L. etc., A taxonomy of data models, in Systems for Large Data Bases, Lockerman, P. C. and Neuhold, E. J. ed., North-Holland, 1976.

[LIN 65]
Lin, S. Computer solution of the traveling salesman problem, Bell System Tech. J., 44, 2245-2269, 1965.

[MSA 77]

Morn's, P. and Sagalowicz, D. Managing network access to a distributed database, proc. 2nd Berkeley Workshop, 58-67, 1977.

[MU 81]

Maier, D., and Ullman, J.D. Connections in acyclic hypergraphs. STAN-CS-81-853, Stanford Univ. May 1981.

[NBC 76]

Nahouraii, E., Brooks, L. O., and Cardenas, A. F. An approach to data communication between different GDBMS. Proc. 2nd VLDB, Sep 1976.

[ROG 77]

Rothnie, J. B. and Goodman, N. An overview of the preliminary design of SDD-1, Proc. 2nd Berkeley Workshop, 39-57, 1977.

[RSL 77]

Rosenkrantz, D.J., Stearns, R.E., Lewis, P.M. an analysis of several heuristics for the traveling salesman problem, SIAM J. Computing, Vol 6, NO. 3, sep 1977.

[SAC 77]

Sacerdoti, E. D. Language access to distributed data with error recovery, SRI Tech. Note 140, 1977.

[SAA 73]

Senko, M. E. Data structures and accessing in data base systems. IBM system J. 12(1) 1973.

[SBD 81]

Smith, J. M. etc. MUTIBASE-integrating heterogeneous distributed database systems, AFIPS NCC vol. 50(487-500) 1981.

[SDD 77]

SDDTG of CODASYL systems committee, A stored data definition language for the translation of data. Inform. Systems 2(3) 1977.

[SEN 76]
Senko, M. E. DIAN as a detailed example of the ANSI/SPARC architecture. IFIP-TC2 Working Conf. Jan 1976.

[SHI 79]
Shipman, D. The functional data model and the data language DAPLEX, Proc. SIGMOD Conf. 1979.

[SHL 75]
Shu, N., Housel, B., Lum, V. CONVERT, a high level translation definition language for data conversion. IBM Rep. RJ 1500, 1975.

[SHM 81]
Shmueli, O. The fundamental role of tree schemas in relational query processing, Ph.D. Thesis, TR-16-81, Aiken Comp. Lab. Harvard Univ.

[SHTGL 77]
Shu, N., Housel, B., Taylor, R. W., Ghosh, S. P., Lum, V. EXPRESS: a data extraction, processing and restructuring system. ACM TODS 2(2), Jun 1977.

[SLH 76]
Shu, N., Lum, V., Housel, B. An approach to data migration in computer networks. IBM Rep. RJ 1703 1976.

[SO 75]
Soshani, A. A logical-level approach to data base conversion. Proc. SIGMOD conf. 112-122. 1975.

[SU 76]
Su, S. Y. W. Application program conversion due to data base changes. Proc. 2nd VLDB, 143-157 Sep 1976.

[ULL 80]
Ullman, J. D. Principles of Database Systems, Computer Series Press, 1980.

[WONG 77]
Wong, E. Retrieving Dispersed data in SDD-1: A systems for distributed databases. Proc. 1977

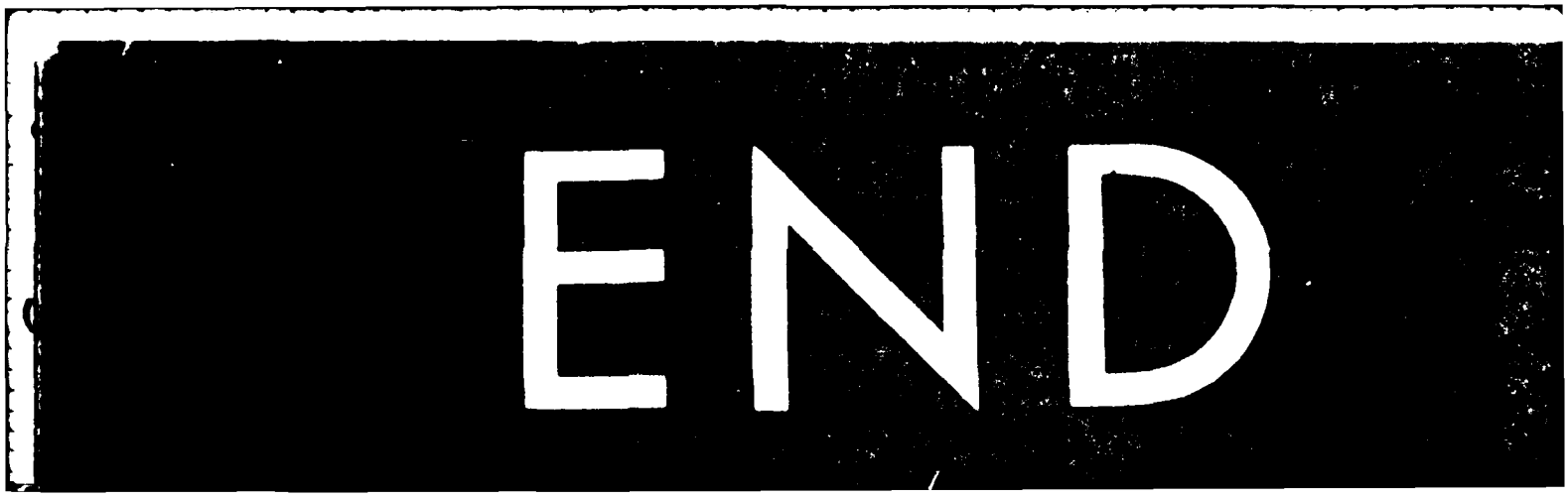
Berkeley Workshop on Distributed Data Management and Computer Network. May 1977.

[YO 79]

Yu, C. T., and Ozsoyoglu, M. Z. An algorithm for tree query membership of a distributed query. Proc. Compsac 79. IEEE Comp. Society Nov 1979.

[ZA 79]

Zaniolo, C. Design of relational views over network schemas, Proc. 1979 ACM SIGMOD Conference, June 1979.


END